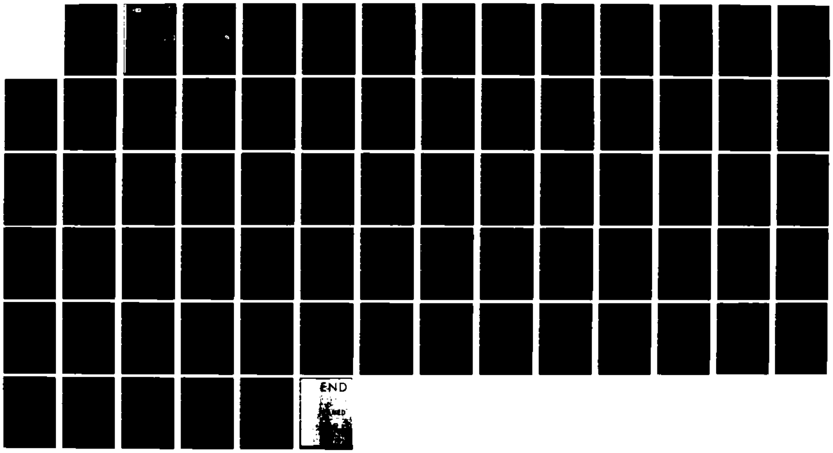
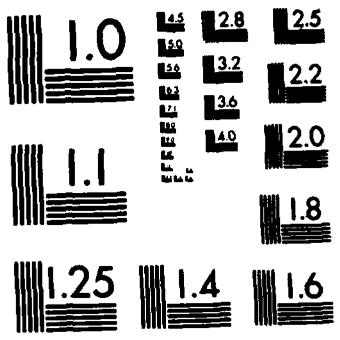
AD-A127 019   FAULT DIAGNOSIS OF NONLINEAR ANALOG CIRCUITS VOLUME III   1/1
FAULT DIAGNOSIS I..(U) PURDUE UNIV LAFAYETTE IN SCHOOL
OF ELECTRICAL ENGINEERING L RAPISARDA ET AL. APR 83
UNCLASSIFIED   N00014-81-K-0323   F/G 9/5   NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Electrical Engineering

FAULT DIAGNOSIS OF NONLINEAR ANALOG CIRCUITS

VOLUME III
FAULT DIAGNOSIS
IN THE TABLEAU CONTEXT
FINAL REPORT

L. Rapisarda
R. DeCarlo

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

April 1983

This work was supported by Office of Naval Research,
Contract No. N00014-81-K-0323.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A127019	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Fault Diagnosis of Nonlinear Analog Circuits, Volumn III, Fault Diagnosis in the Tableau Context.		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Larry Rapisarda, Ray DeCarlo		8. CONTRACT OR GRANT NUMBER(s) N00014-81-K-0323
9. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue University School of Electrical Engineering West Lafayette, Indiana 47907		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, Virginia		12. REPORT DATE April 1983
		13. NUMBER OF PAGES 65
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Fault Diagnosis, Parameter Identification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report details a multifrequency tableau approach for the fault diagnosis of analog linear circuits and systems. The approach utilizes a modified Newton-Raphson solution algorithm to solve a nonlinear (often quadratic) set of tableau equations which characterize the parameters to be identified for the purposes of fault diagnosis. The theory and algorithm are applied to the fault diagnosis of a video amplifier circuit.

Fault Diagnosis in the Tableau Context
Final Report

L. Rapisarda
R. DeCarlo

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

March 1983

This work was supported by Office of Naval Research
Contract No. N00014-81-K-0323

ACKNOWLEDGMENT

The authors would like to thank the Office of Naval Research for the support of this work under ONR Contract No. N00014-81-0323. We would also like to express our gratitude to those responsible for the operation of the Engineering Computer Network at Purdue. The use of this excellent computing resource was vital to our research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

This report finishes the work supported by the Office of Naval Research as cited in the acknowledgement. In particular it indicates the numerical and practical feasibility of the tableau approach to multifrequency fault diagnosis previously introduced in reference [1]. The first chapter presents a 26 component fault diagnosis example based on a linearized model of a video amplifier circuit. The diagnosis of the example circuit depends on the solution of the Tableau Fault Diagnosis Equations. The development of the equations as well as a solution technique are the subject of an earlier report (see reference [1]). Chapter 2 investigates the properties of the tableau fault diagnosis equations under the assumption that the number of simultaneous faults in a system under test is limited. The approach employs multifrequency testing to extract a maximum of information from a given set of test points. The analysis specifically addresses the case in which the number of faulty components exceeds the number of measurement points. The development includes a detailed description of the solution algorithm. Several example problems and solutions conclude the chapter. These include the video amplifier circuit which is used to illustrate the feasibility of the approach. Appendices are included which detail the Fortran code of the computer programs which implement the fault diagnosis techniques for the examples of the report.

Chapter 1
Completion of "Full Diagnosis" Research

1. Introduction

The purpose of this report is the completion of the work presented in [1]. This research concentrated on the "full diagnosis" problem, i.e. the problem of determining system/circuit parameters from multifrequency output measurements under the assumption that all components may be faulty simultaneously. Before discussing new material we summarize the major contributions of the previous report [1].

First is the development of a set of Tableau Fault Diagnosis Equations based on the use of the Component Connection Model (CCM). Due to the nature of the CCM the fault diagnosis equations and the associated Jacobian, utilized in the solution algorithm, have an elegant and sparse structure. Another important characteristic of these equations is that in many cases they are quadratic. The formulation of the diagnosis equations appears in summary in Chapter 2 of this report. The second contribution of [1] is the presentation of a theory of diagnosability for the fault diagnosis equations. This includes a test for diagnosability based on the computation of the rank of the sparse Jacobian. Third is the development of a solution algorithm for the nonlinear Tableau Fault Diagnosis Equations. Of particular interest is a modification which exploits the quadratic nature of the equations and substantially improves the convergence properties of the solution procedure. Finally [1] presents several illustrative examples which include algorithm performance data.

The third section of this chapter completes our work in the application of the fault diagnosis equations to the "full diagnosis" problem with an example which illustrates the feasibility of the Tableau approach for large systems. The second chapter then adapts this work to the assumption that the number of simultaneous faults is limited. Before proceeding to new material we present a review of the CCM and the derivation of the fault diagnosis equations.

2. The Component Connection Model and the Fault Diagnosis Equations.

Let a linear system have N components where the i-th component is characterized by the transfer function $Z_i(s,r_i)$ (s is the Laplace Transform variable and r_i is a parameter which characterizes the component). Denote the i-th component input and output as $a_i(s)$ and $b_i(s)$ respectively. Then

$$b_i(s) = Z_i(s,r_i)a_i(s) \tag{2.1}$$

is the component input/output equation. Next define the composite component input/output vectors:

$$a(s) = \mathrm{col}\,(a_1(s),a_2(s),\ .\ .\ .\ .\ ,a_N(s)) \tag{2.2}$$

$$b(s) = \mathrm{col}\,(b_1(s),b_2(s),\ .\ .\ .\ .\ ,b_N(s)) \tag{2.3}$$

and the composite component transfer function

$$Z(s,r) = \text{block diag}(\ldots,Z_i(s,r_i),\ldots) \tag{2.4}$$

where $r = \mathrm{col}\,(r_1,r_2,\ldots,r_N)$. The composite component input and output vectors are related by

$$b(s) = Z(s,r)a(s) \tag{2.5}$$

The connection laws (e.g. KVL, KCL) are then expressed in terms of the following equations:

$$a(s) = L_{11}b(s)+L_{12}u(s) \tag{2.6}$$

$$y(s) = L_{21}b(s)+L_{22}u(s) \tag{2.7}$$

where u(s) and y(s) are the circuit/system input and output vectors respectively and the L_{ij} are determined by the connections. Equations 2.5, 2.6 and 2.7 form the CCM equations [5] and have a frequency domain tableau formulation:

$$\begin{bmatrix} Z(s,r) & -I \\ -I & L_{11} \end{bmatrix} \begin{bmatrix} a(s) \\ b(s) \end{bmatrix} = \begin{bmatrix} 0 \\ -L_{12}u(s) \end{bmatrix} \tag{2.8a}$$

$$y(s) = L_{21}b(s)+L_{22}u(s) \tag{2.8b}$$

Suppose the circuit/system modeled by equation 2.8 is to be diagnosed; that is the value of r in $Z(s_i,r)$ is sought. Test inputs are applied at q different test input/frequency combinations and the corresponding outputs are measured. Let $u(s_i)$ be the test input vector and $y^M(s_i)$ be the test output vector, for i=1,2,...,q. Because the circuit/system is linear all components of $u(s_i)$ and $y^M(s_i)$ are phasors. It is possible to construct a set of fault diagnosis equations of the form [1] :

$$[Z(s_i,r)\,|\,-V]\begin{bmatrix} L_{11}V\underline{\alpha}_i+a_o(s_i) \\ \underline{\alpha}_i \end{bmatrix} = b_o(s_i) \tag{2.9}$$

for i=1,2,...,q , where

(1) q is the n[illegible] of [illegible] input/frequency combinations,

(2) $b_o(s_i) = L_{21}^{-R}[y^M(s_i) - L_{22}u(s_i)]$,
(3) L_{21}^{-R} is any right inverse of L_{21},
(4) $a_o(s_i) = L_{11}b_o(s_i) + L_{12}u(s_i)$,
(5) V is a matrix whose columns span the null space of L_{21},
(6) r is the unknown parameter vector, and
(7) $\underline{\alpha}_i$ is a vector of auxiliary unknowns which characterizes the ambiguity in the solution for r at any single frequency, s_i.

In particular if

$$f_i(r) \triangleq [Z(s_i,r) \,|\, -V] \tag{2.10}$$

$$g_i(\underline{\alpha}_i) \triangleq \begin{bmatrix} L_{11}V\underline{\alpha}_i + a_o(s_i) \\ \underline{\alpha}_i \end{bmatrix} \tag{2.11}$$

$$\beta_i \triangleq b_o(s_i) \tag{2.12}$$

$$x \triangleq \text{xcol}\,[\underline{\alpha}_1, \underline{\alpha}_2, \ldots, \underline{\alpha}_q, r] \tag{2.13}$$

then the fault diagnosis equations have the equivalent form

$$F(x) = \begin{bmatrix} f_1(r)g_1(\underline{\alpha}_1) - \beta_1 \\ \vdots \\ f_q(r)g_q(\underline{\alpha}_q) - \beta_q \end{bmatrix} = \Theta \tag{2.14}$$

where Θ is the zero vector. Using a Newton-Raphson scheme, one iteratively solves[1] for the solution, say x^*, via

$$J_F(x^k)[x^{k+1} - x^k] = -F(x^k) \tag{2.15}$$

where x^k is the k-th estimate of the solution to equation 2.14 and $J_F(.)$ is the Jacobian of $F(.)$. Because of the product structure of equation 2.13 and the sparsity inherent in 2.9, the Jacobian is both elegant and sparse. Specifically

$$J_F(x) = \begin{bmatrix} f_1(r)\frac{\partial g_1}{\partial \underline{\alpha}_1}(\underline{\alpha}_1) & 0 & . & . & \frac{\partial f_1}{\partial r}(r)g_1(\underline{\alpha}_1) \\ 0 & f_2(r)\frac{\partial g_2}{\partial \underline{\alpha}_2}(\underline{\alpha}_2) & . & . & \frac{\partial f_2}{\partial r}(r)g_2(\underline{\alpha}_2) \\ . & 0 & . & 0 & . \\ 0 & . & . & f_q(r)\frac{\partial g_q}{\partial \underline{\alpha}_q}(\underline{\alpha}_q) & \frac{\partial f_q}{\partial r}(r)g_q(\underline{\alpha}_q) \end{bmatrix} \tag{2.16}$$

3. An Example

To illustrate the use of the fault diagnosis equations developed in [1] consider Figure 1 which is the high-frequency AC equivalent circuit of a four transistor video amplifier. This circuit is based on the example circuit given in Fig. 7 of reference [9]. In addition to the ground node we consider the nodes labeled A through F as accessible since these correspond to inputs, outputs, or power supply connections on the original circuit. These accessible points are used to provide the four inputs and six outputs shown in Figure 1. The nonzero entries of the sparse set of connection matrices, L_{11}, L_{12} and L_{21}, appear in Tables 1 through 3. All entries of L_{22} are zero.

Table 1.
Nonzero entries of L_{11} (26×26).

row,column	value	row,column	value	row,column	value
1,2	-1	1,4	-1	1,5	-1
2,1	1	2,3	-1	3,2	1
3,4	1	3,6	1	4,1	1
4,3	-1	5,1	1	5,7	-1
5,26	-1	6,1	1	6,3	-1
7,5	1	7,6	-1	7,8	-1
7,10	-1	7,11	-1	8,7	1
8,9	-1	8,26	1	9,8	1
9,10	1	9,12	1	9,13	-1
9,20	-1	10,7	1	10,9	-1
10,26	1	11,7	1	12,7	1
12,9	-1	12,26	1	13,9	1
13,14	-1	13,15	-1	14,13	1
14,16	-1	14,17	-1	15,13	1
15,17	-1	15,18	1	16,14	1
17,14	1	17,15	1	17,19	-1
17,26	-1	18,14	1	19,17	1
19,18	-1	20,9	1	20,21	-1
20,22	-1	21,20	1	21,23	-1
21,24	-1	22,20	1	22,24	-1
22,25	1	23,21	1	24,21	1
24,22	1	24,26	-1	25,21	1
26,5	1	26,6	-1	26,8	-1
26,10	-1	26,12	-1	26,17	1
26,18	-1	26,24	1	26,25	-1

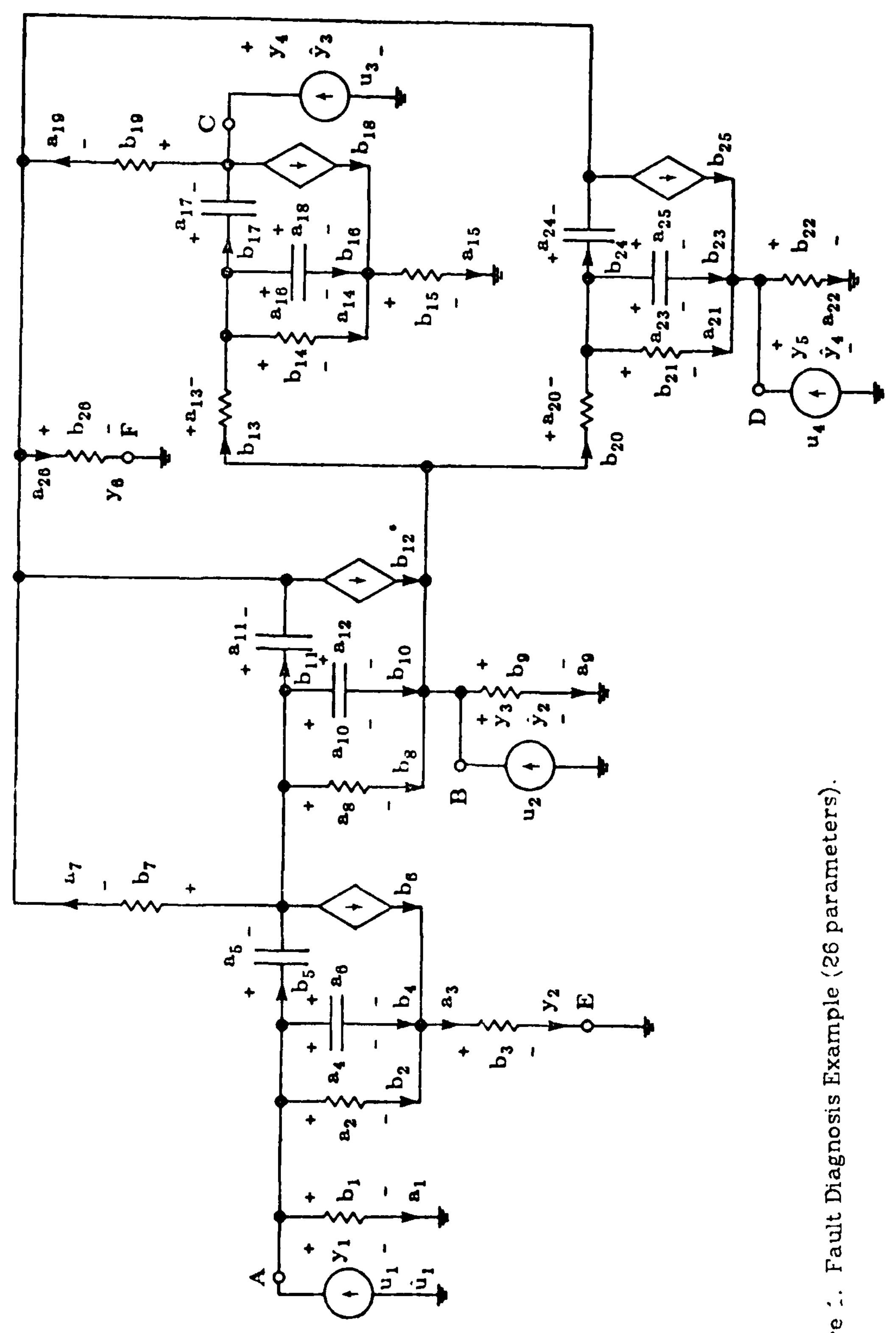


Figure 1. Fault Diagnosis Example (26 parameters).

Table 2.
Nonzero entries of L_{12} (26×4).

row,column	value
1,1	1
9,2	1
19,3	1
22,4	1

Table 3.
Nonzero entries of L_{21} (6×26).

row,column	value	row,column	value
1,1	1	2,2	1
2,4	1	2,6	1
3,9	1	4,19	1
4,26	1	5,22	1
6,5	1	6,6	-1
6,8	-1	6,10	-1
6,12	-1	6,17	1
6,18	-1	6,24	1
6,25	-1		

The computation of L_{21}^{-R} and V was performed via the IMSL routines LGINF and LSVDF respectively [8]. These too are spare matrices with their nonzero entries given in Tables 4 and 5.

Table 4.
Nonzero entries of L_{21}^{-R} (26×6).

row,column	value	row,column	value
1,1	1	2,2	0.346154
2,6	0.0384615	4,2	0.346154
4,6	0.0384615	5,2	0.0384615
5,6	0.115385	6,2	0.307692
6,6	-0.0769231	8,2	-0.0384615
8,6	-0.115385	9,3	1
10,2	-0.0384615	10,6	-0.115385
12,2	-0.0384615	12,6	-0.115385
17,2	0.0384615	17,6	0.115385
18,2	-0.0384615	18,6	-0.115385
19,4	0.5	22,5	1
24,2	0.0384615	24,6	0.115385
25,2	-0.0384615	25,6	-0.115385
26,4	0.5		

Table 5.
Nonzero entries of V (26×20).

row,column	value	row,column	value	row,column	value
2,1	0.808608	3,11	1	4,1	-0.428086
4,2	-0.105202	4,3	-0.105202	4,4	-0.105202
4,5	0.0743889	4,6	-0.105202	4,7	0.105202
4,8	-0.105202	4,9	-0.62699	4,10	0.0743889
5,1	-0.0475652	5,2	0.109844	5,3	0.109844
5,4	0.109844	5,5	0.629435	5,6	0.109844
5,7	-0.109844	5,8	0.109844	5,9	0.132594
5,10	0.629435	6,1	-0.380521	6,2	0.105202
6,3	0.105202	6,4	0.105202	6,5	-0.0743889
6,6	0.105202	6,7	-0.105202	6,8	0.105202
6,9	0.62699	6,10	-0.0743889	7,12	1
8,1	0.0475652	8,2	0.890156	8,3	-0.109844
8,4	-0.109844	8,5	0.0776715	8,6	-0.109844
8,7	0.109844	8,8	-0.109844	8,9	-0.132594
8,10	0.0776715	10,1	0.0475652	10,2	-0.109844
10,3	-0.109844	10,4	0.890156	10,5	0.0776715
10,6	-0.109844	10,7	0.109844	10,8	-0.109844
10,9	-0.132594	10,10	0.0776715	11,13	1
12,1	0.0475652	12,2	-0.109844	12,3	-0.109844
12,4	-0.109844	12,5	0.0776715	12,6	0.890156
12,7	0.109844	12,8	-0.109844	12,9	-0.132594
12,10	0.0776715	13,14	1	14,15	1
15,16	1	16,17	1	17,1	-0.0475652
17,2	0.336293	17,3	0.336293	17,4	0.336293
17,5	-0.237795	17,6	0.336293	17,7	-0.336293
17,8	0.336293	17,9	-0.301165	17,10	-0.237795
18,1	0.0475652	18,2	-0.109844	18,3	0.890158
18,4	-0.109844	18,5	0.0776715	18,6	-0.109844
18,7	0.109844	18,8	-0.109844	18,9	-0.132594
18,10	0.0776715	19,5	0.5	19,10	-0.5
20,18	1	21,19	1	23,20	1
24,1	-0.0475652	24,2	0.109844	24,3	0.109844
24,4	0.109844	24,5	-0.0776715	24,6	0.109844
24,7	0.890156	24,8	0.109844	24,9	0.132594
24,10	-0.0776715	25,1	0.0475652	25,2	-0.109844
25,3	-0.109844	25,4	-0.109844	25,5	0.0776715
25,6	-0.109844	25,7	0.109844	25,8	0.890156
25,9	-0.132594	25,10	0.0776715	26,5	-0.5
26,10	0.5				

The component transfer functions are: $Z_i(s,r_i) = r_i s$ for i=4,5,10,11,16,17,23,24 and $Z_i(s,r_i) = r_i$ for the remaining i.

The nominal component values appear in Table 8 which includes the solution results. Note that the component values are scaled to improve the numerical condition of the problem. The impedance scale factor is 10^2 and the frequency scale factor is 10^7.

For this example M=N=26 and p=20. Using theorem 6.1 the minimum number of input/frequency combinations is q=3. For this value of q we found no set of inputs and frequencies for which the circuit would meet the diagnosability condition of Theorem 4.2. Increasing q to 4 we find that the following inputs and frequencies render the circuit diagnosable:

$$u(s_1) = u(j.05) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.1a}$$

$$u(s_2) = u(j.2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{7.1b}$$

$$u(s_3) = u(j.1) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{7.1c}$$

$$u(s_4) = u(j1.) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{7.1d}$$

The nominal values for the $\underline{\alpha}_i$ to be used along with the nominal parameters as the first solution estimate are in Tables 6 and 7.

Table 6.
Nominal values for $\underline{\alpha}_1$ and $\underline{\alpha}_2$.

$\underline{\alpha}_1$		$\underline{\alpha}_2$	
real part	imag. part	real part	imag. part
−.38813E−01	−.10576E−01	−.35685E−01	−.22477E−02
.10873E+00	.11199E−01	.79972E−01	−.16134E−01
.23505E+00	−.23526E+00	.15327E+00	−.11376E+00
.25592E+00	.20359E+00	−.18606E+00	−.36613E+00
.90589E+00	−.17671E+00	−.37858E+00	−.28573E−01
.84608E+00	−.60119E+00	−.26385E+00	.25068E+00
.31807E+00	−.88404E−01	.23297E−01	−.28580E−01
.78337E−01	−.74524E−01	.10206E+00	−.43404E−01
−.51649E−01	.16461E+00	.54176E−01	.39788E−01
−.10085E−01	.64373E+00	.15078E+00	.52261E−01
.29313E+01	−.42463E+01	.83646E+00	−.55977E−01
.12641E+01	−.19982E+01	.35621E+00	−.17659E+00
.49954E+00	.31602E+00	.17659E+00	.35621E+00
.31785E+00	−.23551E+00	.16082E+00	−.29919E−01
.13377E−01	−.25264E−01	.69824E−02	−.94932E−02
.39654E+00	−.44345E+00	.33091E+00	−.52113E−01
.63160E−01	.33443E−01	.94932E−01	.69824E−01
.44210E+00	−.77673E−01	.13150E+00	−.29041E−01
−.22943E−02	−.91910E−02	.18614E−02	−.24575E−02
.22977E−01	−.57358E−02	.24575E−01	.18614E−01

Table 7.
Nominal values for $\underline{\alpha}_3$ and $\underline{\alpha}_4$.

$\underline{\alpha}_3$		$\underline{\alpha}_4$	
real part	imag. part	real part	imag. part
.21705E−01	.83072E−03	−.37897E−01	−.54919E−03
−.14510E+00	−.54401E−01	.71877E−01	−.54767E−02
.25918E+00	−.21989E+00	.77660E−01	−.27958E−01
−.18065E+00	−.16939E+00	.17227E+00	.88366E−01
.12023E+01	−.33955E+00	−.52193E+00	−.52442E−03
−.37128E+00	.20523E−01	.90419E−01	−.25391E−01
.20897E+00	.82907E−01	.80488E+00	.26640E+00
−.17104E+00	−.54508E−01	.31594E−01	.17439E+00
.11877E+00	.10451E+00	.10015E+00	.72395E−02
−.99168E+00	.42984E+00	.34497E+00	−.90537E−03
−.77921E−01	−.93654E−01	.53431E+00	.44778E−01
.25116E+00	−.79270E−01	−.19795E−01	.30812E−01
.39635E−01	.12558E+00	−.15406E+00	−.98974E−01
−.32688E+00	−.42537E−01	.88285E−01	.72916E−02
.40200E−01	−.16473E−01	.59702E−03	−.22681E−02
.97677E+00	.69241E−01	.23887E+00	.13887E−01
.82365E−01	.20100E+00	.11340E+00	.29851E−01
.65552E−01	.13644E−01	−.22168E−01	.62441E−01
−.28225E−02	.64917E−04	−.40096E−02	.17967E−01
−.32459E−03	−.14112E−01	−.89834E+00	−.20048E+00

We next establish a set of "actual" parameter values which are to be determined from the measurement data. These values likewise appear in Table 8. The simulated measurements corresponding to the actual parameters and the inputs of equation 7.1 are:

$$y^{M}(j.05) = \begin{bmatrix} .30131E+01 \\ .54763E-01 \\ .13175E+01 \\ .10894E+01 \\ .23035E-01 \\ .36800E+00 \end{bmatrix} + j \begin{bmatrix} -.41295E+01 \\ -.74921E-01 \\ -.10620E+01 \\ .35500E+00 \\ -.77400E+00 \\ .78442E+00 \end{bmatrix} \tag{7.2a}$$

$$y^{M}(j.2) = \begin{bmatrix} .76267E+00 \\ .13862E-01 \\ .79305E+00 \\ .37143E+00 \\ .42976E+00 \\ .66459E+00 \end{bmatrix} + j \begin{bmatrix} -.30754E-01 \\ -.54854E-03 \\ -.14179E+00 \\ -.64531E-02 \\ -.40539E-01 \\ .41443E-01 \end{bmatrix} \tag{7.2b}$$

$$y^{M}(j.1) = \begin{bmatrix} -.10196E+00 \\ -.18504E-02 \\ -.10865E+00 \\ .17885E+01 \\ -.26883E+00 \\ -.45267E+00 \end{bmatrix} + j \begin{bmatrix} -.94506E-01 \\ -.17186E-02 \\ -.78825E-01 \\ -.75712E+00 \\ -.11197E+00 \\ -.61275E-02 \end{bmatrix} \tag{7.2c}$$

$$y^{M}(j1.) = \begin{bmatrix} .50661E+00 \\ .92108E-02 \\ .50637E+00 \\ .22685E+00 \\ .54900E+00 \\ .74654E+00 \end{bmatrix} + j \begin{bmatrix} .38092E-01 \\ .69435E-03 \\ .27279E-01 \\ .35169E-02 \\ -.17301E+00 \\ .28013E-02 \end{bmatrix} \quad (7.2d)$$

Table 8 summarizes the results of the solution algorithms. The first solution used the usual Newton-Raphson iteration step while the second solution used the modified algorithm discussed in Chapter 5 of [1]. Both solutions employed a FORTRAN program compiled and executed on a VAX-11/780 with single precision arithmetic. Use of the usual or modified iteration is a program option. This program is included as Appendix A of this report. Although both solution methods converged for this example the modified algorithm required 7 iterations (17 min.) to converge compared to 17 iterations (36.8 min.) for the Newton-Raphson algorithm. The execute times for the programs are somewhat long due to the fact that sparse matrix techniques were not used. The Jacobian for this example is a 208×186 matrix with 5.4% of its elements nonzero.

Table 8.
Parameter values and solution results.

component	units	nominal	actual	solution 1	solution 2
1	ohm	12	11	11	11
2	mho	0.1	0.11	0.109	0.109
3	ohm	56.7	55	55	55
4	farad	50	100	100.2	100.2
5	farad	5	4.5	4.5	4.5
6	mho	10	9	9.016	9.017
7	ohm	30	37	37	37
8	mho	0.1	0.15	0.1498	0.1498
9	ohm	10	8	8	8
10	farad	50	65	65	65
11	farad	5	6.2	6.2	6.2
12	mho	10	14	14	14
13	mho	0.3	0.22	0.22	0.22
14	ohm	10	11	11.03	11.03
15	ohm	2	2.5	2.5	2.5
16	farad	50	45	45.04	45.04
17	farad	5	4.8	4.8	4.8
18	mho	10	9	9.007	9.007
19	ohm	10	11	11	11
20	mho	0.3	0.33	0.33	0.33
21	ohm	10	11	11.23	11.23
22	ohm	10	12	12	12
23	farad	50	49	49	49
24	farad	5	6	6	6
25	mho	10	3.8	3.798	3.798
26	ohm	0.78	0.7	0.7	0.7

4. Summary

The preceding example completes the Tableau Fault Diagnosis Equations documented in [1] by demonstrating the feasibility of their use for a large system. These equations are easily computed, requiring no matrix inversions in the their construction or in the construction of the associated Jacobian. The polynomial order of the Tableau Fault Diagnosis Equations is dependent on the component characteristics and not the size of the system. For many systems this means that the equations are quadratic, a fact which has been exploited to improve the convergence properties of the solution algorithm. Finally a substantial improvement in the efficiency of the solution algorithm is possible upon the application of sparse matrix techniques.

Chapter 2
Fault Diagnosis in the Tableau Context with the Assumption of Limited Simultaneous Faults

1. Introduction

The discussion of the previous report [1] presumed that ALL parameters of a circuit/system could differ significantly from their nominal values. This assumption is appropriate when there is some form of interdependence among the parameters. An example of this is the parameter set of a transistor model. A transistor functioning abnormally could cause all the parameters of a linearized model to change significantly. However parameter failures in many circuits/systems are often statistically independent. In this case the likelihood of more than a few simultaneous failures is extremely small. The fact that most parameters are at or near their nominal values represents information useful to the solution process and leads to the following objective for this chapter:

> Utilize the Tableau fault diagnosis equations developed in [1] to solve for the circuit/system parameter vector, r, given i) the measurement data and ii) the assumption that at most n_f of the N parameters differ from nominal where the integer, n_f, denotes the "number of assumed faults".

Several compelling motivations underly this recasting of the fault diagnosis problem. First it simplifies the calculations since the solution algorithm deals with fewer equations and unknowns. In fact there are recent well documented circumstances [2-4] in which the computations proceed via linear methods. More important than the simplification of the diagnosis equations is the fact that the fault diagnosis process requires fewer test points when the assumption of a limited number of faults is valid. Call the number of test points (outputs) n_o. Each of the n_o test points is a source of information which in composite should suffice to determine the parameters. Intuitively, the amount of information necessary is proportional to the dimension of the parameter space. Since the limited fault assumption forces the solution to lie in a lower dimension subspace of the parameter space, it follows that its use in the diagnosis process should require fewer test points than a complete diagnosis (i.e. no limited parameter failure assumption). Keep in mind however that a tradeoff arises between the reduction of test points and the ease of solving the resulting equations. For a given value of n_f as the number of test points, n_o, decreases the difficulty of solving the fault diagnosis equations should increase.

Current research in the literature has concentrated on simplifying the fault diagnosis calculations. For example Huang, Lin and Liu [2] have developed a node-fault approach which solves for the change in the branch admittances of a linear network using voltage measurements at a set of "accessible" nodes in the network. If the number of accessible nodes is m+1 (including the reference node) and the number of faults is k, this approach requires that $k<m$ for the fault to be uniquely determined using linear techniques. (Note: For this approach the number m is analogous to the number n_0 in the Tableau approach and k is analogous to n_f.) For any set of k faulty branches where $k<m$, the set of possible values for the network response lies in a k-dimensional linear subspace of R^m. If a measurement does not lie in any such subspace then $k \geq m$ and it is not possible to determine the fault uniquely. If on the other hand the measurement does lie in a subspace corresponding to a particular k-fault then the branch admittances corresponding to that fault are solvable.

Likewise, Saeks et al. [3] have introduced a fault diagnosis algorithm in the limited failure context based on the CCM. The authors recognized the excessive computational cost associated with solving a set of fault diagnosis equations for every possible fault combination and their approach circumvents this problem. Their algorithm, applicable to nonlinear as well as linear networks, has the following structure:

i) Partition the components into two groups. Assume that the group 1 components are "good". Using the measurement data and the characteristics of the group 1 components determine the inputs and outputs of the group 2 components which would give rise to the measurement data.

ii) Test each component in group 2 by determining if the component outputs and inputs are consistent with the nominal component characteristic. If the inputs and outputs of component i are consistent with its nominal characteristic then component i is assumed good, if not then no decision is possible.

iii) Repartition the components including all components found to be good in group 1. Go to step ii and continue the process until group 1 consists entirely of good components.

Several remarks concerning this procedure are relevant to the development of the algorithm in this paper. First the decision rule used in [3] to establish the identity of the good components is exact for single component failures but for multiple faults the decision rule must be based on the assumption that the effects of multiple faults cannot cancel. This assumption is reasonable when the "good" components are represented by parameters which are exactly nominal but may break down when actual values of the good parameters are randomly spread around nominal due to production variations. Second, the algorithm in [3] requires that the number of test points for a linear system be sufficient to permit the computation of the inputs and outputs of the components under test with single frequency test data. Third, use of the multiple fault decision rule requires that the number of faults be strictly less than the number of components under test. (Note: These two requirements are equivalent to the restriction in the Huang, Lin and Liu approach [2] that $k<m$.) Finally although this method employs the CCM it differs fundamentally from the Tableau Approach presented in this paper.

A final example of the use of the limited fault assumption is that of Biernacki and Bandler who developed an approach to multiple fault location for linear networks. Here the faults are modeled as loads applied to an invariant network which represents the system in its nominal state [4]. Their fault diagnosis equations build on voltage measurements taken at a single test frequency. The identification of the faults depends upon checking the consistency of a set of linear equations. Like the other approaches the linearity of the fault diagnosis equations requires that the number of voltages measured be greater than the number of simultaneous faults to be located. (Note: As for the previous approach this assumption is equivalent to the restriction in the Huang, Lin and Liu approach [2] that $k<m$ or equivalently in the Tableau context that $n_f<n_o$.)

All the above approaches share the constraint that $n_f<n_o$. The algorithm for limited fault analysis developed in this chapter does not require that the number of faults, n_f, be less than the number of measurement points, n_o. Philosophically, the idea is to use the multifrequency techniques developed thus far to "squeeze" as much information as possible out of a given set of test points. The next section presents several simple examples in the context of the Tableau Fault Diagnosis Equations applied to the limited fault assumption. These examples will clearly illustrate how the relationship between n_o and n_f affects the approach to the solution.

2. Motivational Examples

The purpose of this section is to illustrate the properties of the Tableau Fault Diagnosis Equations under the assumption that the number of parameters which deviate significantly from nominal is limited. The examples are designed to highlight the differences which occur as the relation between n_f and n_o changes and to serve as background to the development of the solution algorithm.

Consider the example shown in Figure 2. The CCM equations for this example are:

$$\begin{bmatrix} b_1(s) \\ b_2(s) \\ b_3(s) \\ b_4(s) \end{bmatrix} = \begin{bmatrix} r_1 & & & \\ & r_2/s & 0 & \\ & 0 & r_3 & \\ & & & r_4 \end{bmatrix} \begin{bmatrix} a_1(s) \\ a_2(s) \\ a_3(s) \\ a_4(s) \end{bmatrix} \tag{2.1a}$$

$$\begin{bmatrix} a_1(s) \\ a_2(s) \\ a_3(s) \\ a_4(s) \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_1(s) \\ b_2(s) \\ b_3(s) \\ b_4(s) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(s) \tag{2.1b}$$

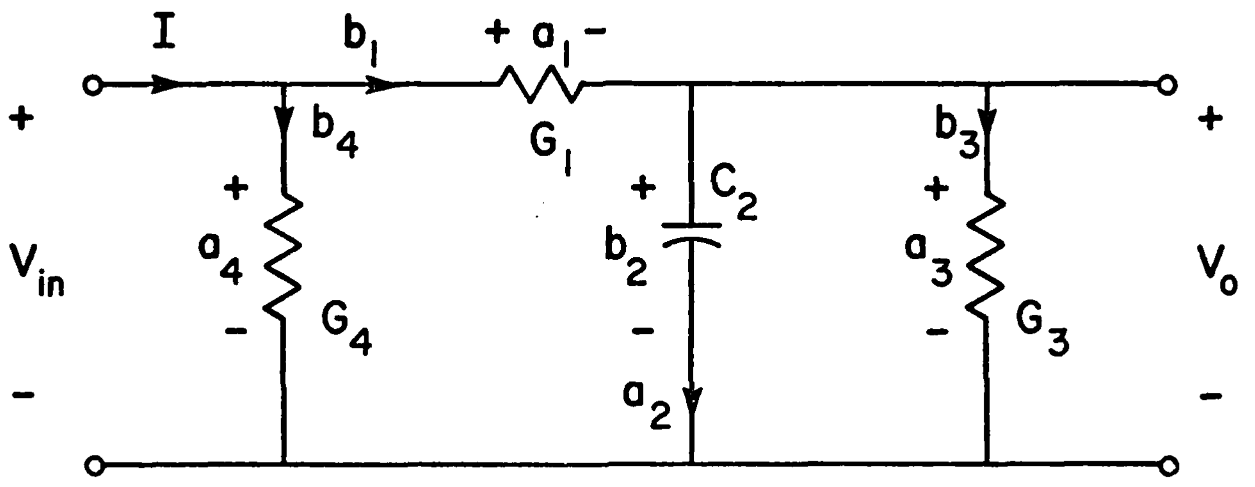


Figure 2. Circuit for Motivational Examples.

$$\begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_1(s) \\ b_2(s) \\ b_3(s) \\ b_4(s) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u(s) \tag{2.1c}$$

where

$$r = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} G_1 \\ 1/C_2 \\ G_3 \\ G_4 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \begin{bmatrix} I \\ V_o \end{bmatrix}$$

The right inverse and null space basis for L_{21} are:

$$L_{21}^{-R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad V = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ -1 & 0 \end{bmatrix}$$

The nominal value for the parameter vector, r_o is r_o = col[1 1 1 1] and n_o=2.

Case 1: n_f=1 (Component 1 faulty)

Assume that only one parameter value differs from nominal and that measurements occur for a single real test frequency, $s=s_1$. Recall that the fault diagnosis equation is:

$$\begin{bmatrix} Z(s_1,r) \mid -V \end{bmatrix} \begin{bmatrix} L_{11}V\underline{\alpha}_1 + a_o(s_1) \\ \text{-------} \\ \underline{\alpha}_1 \end{bmatrix} - b_o(s_1) = \Theta \tag{2.2}$$

where:

i. $b_o(s_1) = L_{12}^{-R}\left[y^M(s_1) - L_{22}u(s_1)\right]$;
ii. $a_o(s_1) = L_{11}b_o(s_1) + L_{12}u(s_1)$;
iii. The columns of V span null[L_{21}];
iv. L_{21}^{-R} is any right inverse of L_{21};
v. $y^M(s_1)$ is the measurement (an n_o dimensional vector);
vi. The dimension of null[L_{21}] is p;
vii. $\underline{\alpha}_1 = \text{col}(\alpha_1(s_1), \alpha_2(s_1), \ldots, \alpha_p(s_1))$.

Next, rearrange the fault diagnosis equation 2.2 to produce:

$$\left[V - Z(s_1,r)L_{11}V\right]\underline{\alpha}_1 = Z(s_1,r)a_o(s_1) - b_o(s_1) \tag{2.3}$$

Substituting the information for this example into equation (2.3) yields:

$$\begin{bmatrix} 1 & 0 \\ -r_2/s_1 & r_2/s_1 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1(s_1) \\ \alpha_2(s_1) \end{bmatrix} = \begin{bmatrix} r_1u(s_1) \\ 0 \\ 0 \\ r_4u(s_1) \end{bmatrix} + \begin{bmatrix} -1 & -r_1 \\ r_2/s_1 & -1 \\ 0 & r_3 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1^M(s_1) \\ y_2^M(s_1) \end{bmatrix} \tag{2.4}$$

Notice the following characteristics of this set of equations:

i. There are 4 equations and 6 unknowns (4 parameters and 2 ambiguity variables);

ii. Each r_i appears only in the i'th equation;

iii. If the nominal numerical values for r replace the variables, the equation set is linear with the α_1 and α_2 as unknowns;

iv. If one parameter is faulty then a solution for α_1 and α_2 must exist upon deleting the equation containing the nominal value of the faulty component.

The above characteristics suggest a solution method. Namely eliminate the i'th equation. If the remaining equations are not consistent then r_i is not faulty (or the single fault assumption is not valid).

To illustrate such a solution method let component 1 be faulty with $r_1 = 2$. Given the test frequency $s_1 = 1$ and input $u(s_1) = 1$ the corresponding test outputs are:

$$\begin{bmatrix} y_1^M(s_1) \\ y_2^M(s_1) \end{bmatrix} = \begin{bmatrix} 2. \\ .5 \end{bmatrix} \tag{2.5}$$

Equation 2.3 becomes:

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \\ .5 \\ 1. \end{bmatrix} \tag{2.6}$$

where $\alpha_1 = \alpha_1(1)$ and $\alpha_2 = \alpha_2(1)$. The results of the successive elimination of each of the rows of equation 2.6 appear in summary form in Table 9.

Table 9.
Summary of Consistency Check (Component 1 Faulty).

Row Elim.	Resulting Equation	Solution
1	$\begin{bmatrix}-1 & 1\\ 0 & 1\\ -1 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix}=\begin{bmatrix}1.5\\ .5\\ 1.\end{bmatrix}$	$\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix}=\begin{bmatrix}-1\\ .5\end{bmatrix}$
2	$\begin{bmatrix}1 & 0\\ 0 & 1\\ -1 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix}=\begin{bmatrix}-1.5\\ .5\\ 1.\end{bmatrix}$	Inconsistent
3	$\begin{bmatrix}1 & 0\\ -1 & 1\\ -1 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix}=\begin{bmatrix}-1.5\\ 1.5\\ 1.\end{bmatrix}$	Inconsistent
4	$\begin{bmatrix}1 & 0\\ -1 & 1\\ 0 & 1\end{bmatrix}\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix}=\begin{bmatrix}-1.5\\ 1.5\\ .5\end{bmatrix}$	Inconsistent

The data in Table 9 clearly indicates that the only possible single fault for the given measurement data is component 1. To determine the parameter values simply substitute the α_i values into 2.7a and 2.7b to compute actual component inputs and outputs:

$$b(1) = b_o(1) + V\underline{\alpha}_1 \tag{2.7a}$$

$$a(1) = a_o(1) + L_{11}V\underline{\alpha}_1 \tag{2.7b}$$

Then use the composite component transfer function matrix equation b(1) = Z(1,r)a(1) to compute r. For this example equation 2.7 yields:

$$b(1) = \begin{bmatrix}1\\ .5\\ .5\\ 1\end{bmatrix} \quad \text{and} \quad a(1) = \begin{bmatrix}.5\\ .5\\ .5\\ 1\end{bmatrix}$$

and the parameter values are $r_1 = 2$ and $r_2 = r_3 = r_4 = 1$ as expected.

Case 2: $n_f=1$ **(Component 2 faulty)**

The solution to a limited fault problem is not necessarily unique. To demonstrate this repeat the example with component 2 faulty. Let $r_2 = 2$ with the remainder nominal. The test outputs for this case given $s_1 = 1$ and $u(s_1) = 1$ are:

$$\begin{bmatrix}y_1^M(s_1)\\ y_2^M(s_1)\end{bmatrix} = \begin{bmatrix}1.6\\ .4\end{bmatrix} \tag{2.8}$$

Substitute into equation 2.4 to get:

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1. \\ 1.2 \\ .4 \\ 1. \end{bmatrix} \tag{2.9}$$

Table 10 contains the results.

Table 10.
Summary of Consistency Check (Component 2 Faulty).

Row Elim.	Resulting Equation	Solution
1	$\begin{bmatrix} -1 & 1 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1.2 \\ .4 \\ 1. \end{bmatrix}$	Inconsistent
2	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1. \\ .4 \\ 1. \end{bmatrix}$	$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1 \\ .4 \end{bmatrix}$
3	$\begin{bmatrix} 1 & 0 \\ -1 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1. \\ 1.2 \\ 1. \end{bmatrix}$	$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1 \\ .2 \end{bmatrix}$
4	$\begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \\ .5 \end{bmatrix}$	Inconsistent

The data in Table 10 indicates that there are two possible single faults which satisfy the fault diagnosis equations. The values of α from the second row in the table correspond to parameter values: $r_2 = 2$ with the remainder nominal. Those of the third row correspond to $r_3 = .5$ with the remainder nominal. This illustrates the potential for ambiguity in the solutions. It is easy to see for this simple circuit that the ambiguity is generic; that is, it will always occur for this test arrangement whenever parameters two or three are faulty. Although ambiguities are always possible, it would be convenient to be able to avoid such generic ambiguities. A later section will develop a test to determine if such a problem exists for a given diagnosis situation.

To summarize cases 1 and 2 of the example consider the following characteristics: First, the number of faults, n_f, is strictly less than the number of test points, n_o. Next, each fault generated a new set of equations from the original fault diagnosis equations. When a solution failed to exists for the set corresponding to some fault then that fault was not possible. The existence of solutions for more than one fault possibility may be generic to the circuit/system test point combination. Finally, notice that the solution process proceeded via linear methods. This is possible because $n_f < n_o$. Here the use of measurements at a single test frequency provides sufficient data for the solution process in that the set of measurements for each fault combination must lie in a linear subspace of the entire measurement space. Although the solution to these cases is computationally different from the approaches discussed earlier [2-4] the use of linear methods and its underlying justification is an essential commonality which makes them all philosophically equivalent.

Case 3: $n_f=2$ (Components 1 and 2 faulty)

Consider the possibility that the circuit modeled by equation 2.1 has two simultaneous faults. There are six possible fault combinations, i.e.: r_1,r_2 faulty; r_1,r_3 faulty; etc.; all combinations of four parameters taken two at a time. Consider equation 2.4 under the circumstance of two simultaneous faults. As before there are 4 equations and 6 unknowns. Since a fault combination includes two parameters eliminating the equations which contain an assumed fault while setting the remaining parameters to nominal leaves two equations in two unknowns. This means that a solution for α_1 and α_2 is likely to exist for each possible fault combination.

Suppose that the actual fault combination for this example is components 1 and 2 (r_1 = 2 and r_2 = 2) and that s_1 = 1. The test outputs for this case are:

$$\begin{bmatrix} y_1^M(s_1) \\ y_2^M(s_1) \end{bmatrix} = \begin{bmatrix} 1.86 \\ .57 \end{bmatrix} \tag{2.10}$$

As before use the nominal parameter values in equation 2.4 with the understanding that two of the four equations must be incorrect since two parameters are faulty. Equation 2.4 becomes:

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1.86 \\ .57 \end{bmatrix} = \begin{bmatrix} -1.43 \\ 1.29 \\ .57 \\ 1. \end{bmatrix} \tag{2.11}$$

Table 11 summarizes the computations of the α_i and the corresponding r for each possible fault.

Table 11.
Summary of Parameter Computation at $s_1 = 1$ (Components 1 and 2 Faulty).

Rows Elim.	Resulting Equation	Solution	Parameters
1&2	$\begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}.57\\1.\end{bmatrix}$	$\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-1\\.57\end{bmatrix}$	$r_1=2$ $r_3=1$ $r_2=2$ $r_4=1$
1&3	$\begin{bmatrix}-1 & 1\\-1 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}1.29\\1.\end{bmatrix}$	$\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-1\\.29\end{bmatrix}$	$r_1=2$ $r_3=.5$ $r_2=1$ $r_4=1$
1&4	$\begin{bmatrix}-1 & 1\\0 & 1\end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}1.29\\.57\end{bmatrix}$	$\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-.72\\.57\end{bmatrix}$	$r_1=2.7$ $r_3=1$ $r_2=1$ $r_4=.72$
2&3	$\begin{bmatrix}1 & 0\\-1 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-1.43\\1\end{bmatrix}$	Inconsistent	Inconsistent
2&4	$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-1.43\\.57\end{bmatrix}$	$\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-1.43\\.57\end{bmatrix}$	$r_1=1$ $r_3=1$ $r_2=-4$ $r_4=1.43$
3&4	$\begin{bmatrix}1 & 0\\-1 & 1\end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-1.43\\1.29\end{bmatrix}$	$\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}-1.43\\-.14\end{bmatrix}$	$r_1=1$ $r_3=-.25$ $r_2=1$ $r_4=1.43$

As expected the computation produces inconclusive results. Clearly the resolution of the ambiguity in the data of Table 11 requires additional information. To acquire the needed information repeat the same procedure using a second test frequency (the verification step). Let $s_2 = 2$ for which measurement at the output would yield:

$$\begin{bmatrix}y_1^M(s_2)\\y_2^M(s_2)\end{bmatrix} = \begin{bmatrix}2.\\.5\end{bmatrix} \tag{2.12}$$

As before substitute this information into equation 2.4 to obtain:

$$\begin{bmatrix}1 & 0\\-.5 & .5\\0 & 1\\-1 & 0\end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\end{bmatrix} = \begin{bmatrix}1\\0\\0\\1\end{bmatrix} + \begin{bmatrix}-1 & -1\\.5 & -1\\0 & 1\\0 & 0\end{bmatrix}\begin{bmatrix}2.\\.5\end{bmatrix} = \begin{bmatrix}-1.5\\.5\\.5\\1.\end{bmatrix} \tag{2.13}$$

Table 12 contains the summary of results for the computation at this test frequency.

Table 12.
Summary of Parameter Computation at $s_1 = 2$ (Components 1 and 2 Faulty).

Rows Elim.	Resulting Equation	Solution	Parameters
1&2	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} .5 \\ 1. \end{bmatrix}$	$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1 \\ .5 \end{bmatrix}$	$r_1=2$ $r_3=1$ $r_2=2$ $r_4=1$
1&3	$\begin{bmatrix} -.5 & .5 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} .5 \\ 1. \end{bmatrix}$	$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$	$r_1=2$ $r_3=0$ $r_2=1$ $r_4=1$
1&4	$\begin{bmatrix} -.5 & .5 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} .5 \\ .5 \end{bmatrix}$	$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -.5 \\ .5 \end{bmatrix}$	$r_1=3.$ $r_3=1$ $r_2=1$ $r_4=.5$
2&3	$\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1 \end{bmatrix}$	Inconsistent	Inconsistent
2&4	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ .5 \end{bmatrix}$	$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ .5 \end{bmatrix}$	$r_1=1$ $r_3=1$ $r_2=\infty$ $r_4=1.5$
3&4	$\begin{bmatrix} 1 & 0 \\ -.5 & .5 \end{bmatrix}\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \end{bmatrix}$	$\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -.5 \end{bmatrix}$	$r_1=1$ $r_3=-1$ $r_2=1$ $r_4=1.5$

The final step in this two-fault case is the comparison of the Tables 11 and 12. The only common solution between the two tables is that for the component 1 and 2 fault combination. This fault set therefore is the only one which simultaneously satisfies the equations derived from measurements at both test frequencies.

This case for which the number of faults, n_f, equals the number of test points, n_o, prompts the following observations: In general a single real test frequency is insufficient to resolve the ambiguity. Data from the measurement at a second test frequency is necessary. Since the equations resulting from the different test frequencies are solved separately, linear techniques are still possible. The only significant difference between this case and the previous two lies in the necessity for the verification step (the second test frequency). This step was not necessary in cases 1 and 2 since the equations used to solve for the α's were overspecified due to the fact that $n_f < n_o$.

Case 4: $n_f=2$, $n_o=1$

Finally consider the same example (Figure 2) with one test point instead of two. The CCM equations are the same as equation 2.1 except that the output equation is:

$$y_1(s) = [1\ 0\ 0\ 1]\begin{bmatrix} b_1(s) \\ b_2(s) \\ b_3(s) \\ b_4(s) \end{bmatrix} + [0]\,u(s) \tag{2.14}$$

Accordingly the right inverse and null space basis of L_{21} change to:

$$L_{21}^{-R} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}$$

The fault diagnosis equations at a single test frequency s_i for $u(s_i) = 1$ are:

$$\begin{bmatrix} r_1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & r_2/s_i & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & r_3 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & r_4 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1-\alpha_2(s_i) \\ y_1^M(s_i)+\alpha_1(s_i)-\alpha_2(s_i) \\ \alpha_2(s_i) \\ 1 \\ \alpha_1(s_i) \\ \alpha_2(s_i) \\ \alpha_3(s_i) \end{bmatrix} - \begin{bmatrix} y_1^M(s_i) \\ 0 \\ 0 \\ 0 \end{bmatrix} = \Theta \tag{2.15}$$

For the previous cases $n_0 \geq n_f$. To illustrate the properties of the fault diagnosis equations for $n_f > n_o$ let the number of allowable simultaneous faults be two ($n_f = 2$) and notice that $n_o = 1$. Under these circumstances observe that for equation 2.15, the elimination of two equations corresponding to an assumed fault combination with the remaining two parameters set to nominal leaves two equations with three unknowns. This means that the ambiguity must be resolved by the use of several test frequencies. The resulting equations must be solved simultaneously with the unfortunate consequence that the solution method cannot employ linear techniques.

This case represents a more general limited fault problem than the previous cases since the number of test outputs, n_o, may be smaller than the number of assumed faults, n_f. Since this more general approach is the subject of the remainder of this chapter the actual solution process will appear in a later section.

Current research [2-4] has concentrated on the fault diagnosis problem with the restriction that $n_o \leq n_f$. A motivation for this is that the solution method may use linear techniques as illustrated by all but the last case. Unfortunately the restriction to linear methods fails to exploit all of the information available at the test points. Thus problems such as case 4 are not solvable via such methods. The remainder of this chapter considers the development of a more general algorithm for limited fault analysis in the CCM context. Here the restriction, $n_o \leq n_f$, will not apply. The method will exploit all the information available at a given set of test points by utilizing multifrequency testing.

3. Introduction to Notation

The purpose of this brief section is to develop the notation to assist in describing the n_f-fault solution algorithm presented in the next section. To this purpose we define the following:

$$\langle i_1, i_2, \ldots, i_{n_f} \rangle \triangleq \text{The fault index}$$

It is an ordered n_f-tuple of positive integers subject to the following:

$$1 \leq i_1 < i_2 < \cdots < i_{n_f} \leq N \tag{3.1}$$

$$\Omega(N,n_f) \equiv \{ \text{The set of all possible } n_f\text{-tuples satisfying 3.1} \}$$

There are $C_{N:n_f}$ elements in $\Omega(N,n_f)$. (Note: $C_{N:n_f}$ denotes the number of combinations of N things taken n_f at a time.)

If $\gamma = <i_1,i_2,\ldots,i_{n_f}>$ then let M_γ be the following matrix with n_f columns:

$$M_\gamma = \left[e_{i_1} | e_{i_2} | \ldots | e_{i_{n_f}} \right] \tag{3.2}$$

where e_i is the N-dimensional unit vector with a 1 in the i-th position and 0's elsewhere. For the fault $\gamma \in \Omega(N,n_f)$ define the fault space P_γ in the following way:

$$P_\gamma = \left\{ r \mid r = r_o + M_\gamma \rho,\ \rho \in R^{n_f} \right\} \tag{3.3}$$

We say that the fault index γ "represents" the parameter r if $r \in P_\gamma$. In other words γ is a specific n_f-fault combination.
Note: Limiting the number of allowable faults to n_f means that there are ***at most*** n_f faults since any fault space with less than n_f faults is contained in some P_γ as defined in 3.3.

Finally let

$$F_\gamma(x_\gamma) \triangleq F(x) \Big|_{r_i = r_{oi} \quad i \notin \gamma} \tag{3.4}$$

where:

(i) $F(x)$ is as defined in equation 2.14 of Chapter 1.
(ii) $x = \text{col}(\underline{\alpha}_1,\underline{\alpha}_2,\ldots,\underline{\alpha}_q,r)$
(iii) $x_\gamma = \text{col}(\underline{\alpha}_1,\underline{\alpha}_2,\ldots,\underline{\alpha}_q,r_{\gamma_1},\ldots,r_{\gamma_{n_f}})$
(iv) γ_i denotes the i'th element of the fault index γ
(v) r_{oi} is the nominal value of the i'th parameter.

If a system fault is known to lie in P_γ then the actual values of the faulty parameters must satisfy $F_\gamma(x_\gamma) = \Theta$. Suppose a system is n_f-fault diagnosable, then there will generally be one fault index, γ, for which $F_\gamma(x_\gamma) = \Theta$.

4. Limited Fault Algorithm

This section has the following objective:

> Given a circuit/system which is n_f-fault diagnosable and which has a fault characterized by a parameter vector, $r \in P_\gamma$, $\gamma \in \Omega(N,n_f)$, develop a solution algorithm which will: i) determine which circuit/system parameters are faulty (find γ); and ii) solve for the values of the faulty parameters, i.e. find $r \in P_\gamma$, $\gamma \in \Omega(N,n_f)$.

The information available to accomplish this objective is the fact that the parameter vector, r, and the associated ambiguity variables, $\underline{\alpha}_i$, $i=1,2,\ldots,q$, which combine to form the composite unknown vector, x, must satisfy $F(x) = \Theta$ AND the constraint that $r \in P$ where $P = \bigcup_{\gamma \in \Omega(N,n_f)} P_\gamma$. The latter constraint makes it impossible to utilize the Newton-Raphson iteration directly. To illustrate the nature of the problem imposed by this constraint consider the following "brute force" approach as a solution method:

Try to solve for x_γ in $F_\gamma(x_\gamma)=\Theta$ for each $\gamma\in\Omega(N,n_f)$. If for a particular γ, a solution fails to exist, γ denotes an incorrect fault combination. If a solution does exist then γ is a potential fault combination. If only one γ yields a solution, the fault is uniquely identified.

The modified Newton-Raphson algorithm developed in [1] is a stable method for testing each F_γ. The number of such equations which must be tested is $C_{N:n_f}$ where

$$C_{N:n_f}=\frac{n!}{m!(n-m)!} \tag{4.1}$$

The major shortcoming of such an approach is that as N gets large with $n_f>2$, the number of possible faults becomes extremely large resulting in excessively long computation time.

The excessive number of fault possibilities for large systems is the main reason for considering the alternative approach described in this section. For clarity we begin with a brief overview of the idea. Recall that the available information for a solution algorithm to determine the fault index, γ, and solve for the parameter vector, r, is:

$$F(x) = \Theta \tag{4.2}$$

and

$$r \in P = \bigcup_{\gamma\in\Omega(N,n_f)} P_\gamma \tag{4.3}$$

Suppose we view equation 4.2 as a set of constraints and reformulate the restriction in 4.3 in the following way:

Define:

$$\varphi(r,\gamma) \triangleq \|r-r_\gamma\|_2 \tag{4.4}$$

where (i) $r\in R^N$, (ii) $r_\gamma\in P_\gamma$, (iii) $\gamma\in\Omega(N,n_f)$, and (iv) $\|\cdot\|_2$ denotes the Euclidean norm. Notice that since $\varphi\geq0$, equation 4.3 is satisfied if we require that $\varphi(r,\gamma)$ be a minimum. In other words φ, the objective function to be minimized, is the distance from the solution, r, to a point in some P_γ. This distance is zero for any solution that can be described by a fault index, $\gamma\in\Omega(N,n_f)$. Thus the problem statement becomes:

$$\underset{r\in R^N \;\; \gamma\in\Omega(N,n_f)}{\text{minimize}} \; \varphi(r,\gamma) \tag{4.5}$$

subject to:

$$F(x) = \Theta \tag{4.6}$$

To develop some insight into the solution process to be developed consider first a solution method for the general nonlinear constrained minimization problem given by Luenberger[6] and known as the "Gradient Projection Method". This method proceeds in the following manner:

(i) Given a starting point use some solution procedure to find a "feasible point" that is a point which satisfies the equality constraints.

(ii) Next update the feasible point by adding to it a component which lies in the space tangent (at the feasible point) to the curve defined by the equality constraints. The direction of the tangent component is the projection of the gradient of the objective function into the tangent space (hence the Gradient Projection Method). The updated point is the value along the tangent projection for which the objective function is minimum.

(iii) Using the updated point as the starting point repeat these two steps until the procedure converges to a solution point.

To apply this type of procedure to the solution of 4.5 and 4.6, modify the second step of the Gradient Projection Method in the following manner:

(i) Let $\Gamma \subset \Omega(N, n_f)$. For each $\gamma \in \Gamma$ compute the tangent space component for which $\varphi(r, \gamma)$ is a minimum.

Note: Although this requires a search as in the direct approach the time required can be kept relatively small if the number of elements in Γ is small.

(ii) To choose the elements of Γ consider the parameter values at the feasible point. Any parameters which are close to nominal (say within 10%) consider good. This will eliminate some fault combinations from consideration thus decreasing the number of elements of Γ relative to the number of elements of $\Omega(N, n_f)$.

(iii) Another criterion which will limit the size of Γ is to consider the component whose parameter value at the feasible point is farthest from nominal as faulty.

We now present the details of the algorithm to solve the limited fault problem. This algorithm has the structure shown in Figure 3. The first step of the algorithm is to choose an initial guess. The best available information about the solution is the nominal values for the circuit/system under test. Therefore let $x^0 = x_o$.

The next step is to find a feasible solution, that with x_o as an initial guess find a point, $x \in R^{pq+N}$, which satisfies $F(x) = \Theta$. Recall that the modified Newton-Raphson iteration step is[1]:

$$J_F(x^k)d^k = -F(x^k) \qquad (4.7)$$

$$x^{k+1} = x^k + \lambda d^k$$

where λ is chosen to satisfy

$$\|F(x^k + \lambda d^k)\|_2^2 < \|F(x^k)\|_2^2$$

$J_F(x^k)$ can be expected to have a nontrivial null space. If it did not then there is

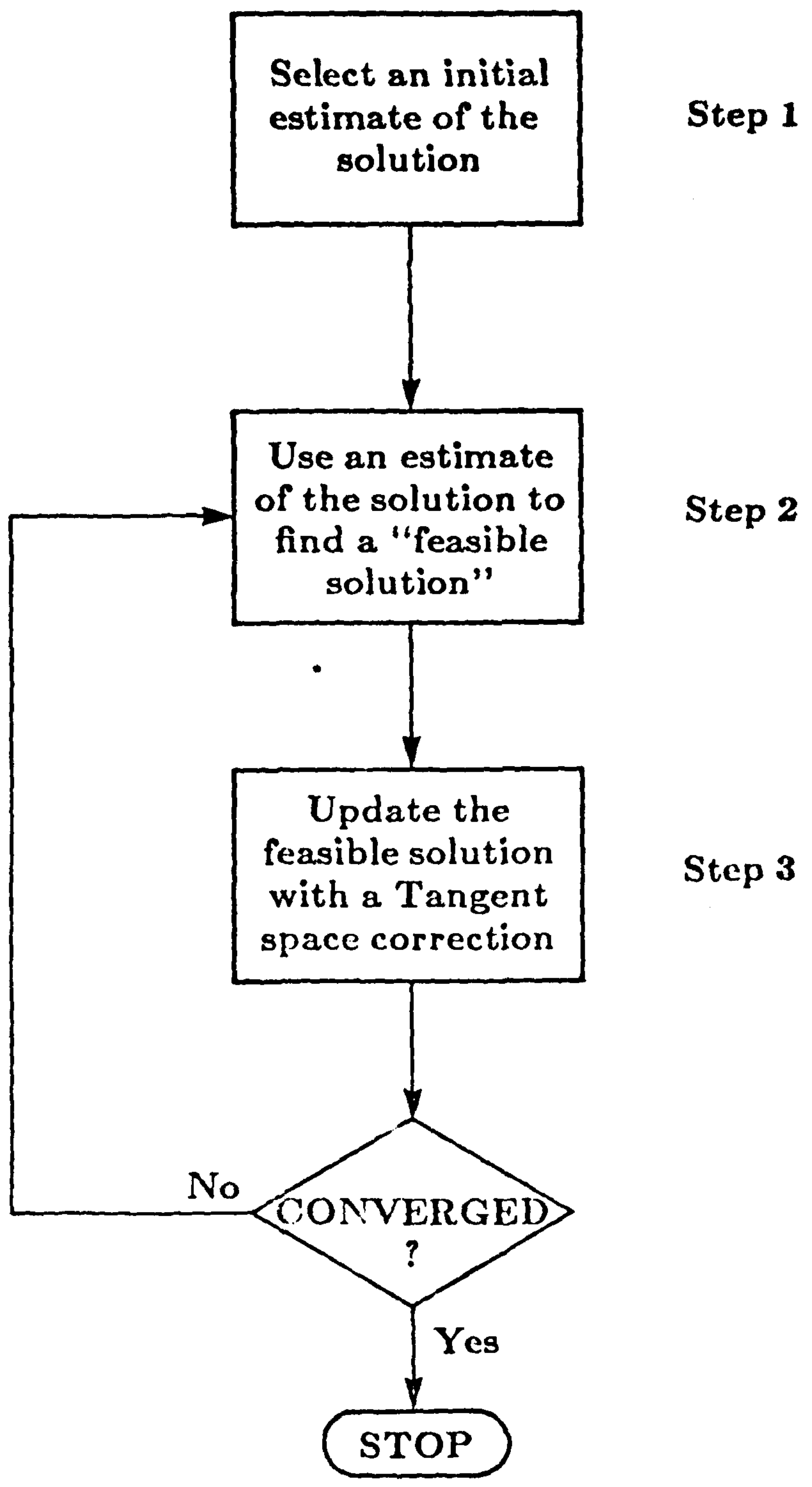

Select an initial estimate of the solution
Step 1
Use an estimate of the solution to find a "feasible solution"
Step 2
Update the feasible solution with a Tangent space correction
Step 3
No
CONVERGED ?
Yes
STOP

Figure 3. Basic Algorithm Structure

sufficient information to find the solution without the limited fault assumption. This means that 4.7 does not have a unique solution. To make the solution to 4.7 unique requires that d^k be the "least squares" solution. This "least squares" solution can theoretically be found via Moore-Penrose pseudo-inverse[7] but in practice the solution is determined using software such as IMSL routine LLBQF[8]. Implemented in this manner the modified Newton-Raphson iteration provides a means of finding a feasible point.

The third step depicted in Figure 3 is the tangent space update of the feasible point to minimize the objective function. To assist in the explanation of the implementation of this step define the following:

$x_{fe} \triangleq$ the feasible solution resulting from Step 2.
$r_{fe} \triangleq$ the parameter part of x_{fe} (i.e. the last N entries of x_{fe}).
$V_F(x_{fe}) \triangleq$ the matrix whose columns span $\mathrm{Null}[J_F(x_{fe})]$.
$V_r(x_{fe}) \triangleq$ the matrix consisting of the last N rows of $V_F(x_{fe})$.
$T_F(x_{fe}) \triangleq$ the tangent space of the surface $F(x) = \Theta$ at the point x_{fe}.
$x_t \triangleq$ a point in $T_F(x_{fe})$.
$r_t \triangleq$ the last N entries of x_t.
$D_\gamma \triangleq$ for $\gamma = <i_1, i_2, \ldots, i_{n_f}>$ this is the identity matrix with rows i_1, i_2 etc. deleted.

Note: D_γ is a matrix operator for the parameter vector which produces a vector consisting of the entries of r whose indices are not contained in γ.

The objective of this step is to find a point in the tangent space which minimizes the function φ. If x_t is a point in $T_F(x_{fe})$ it has the following characterization:

$$x_t = x_{fe} + V_F(x_{fe})\beta \tag{4.8}$$

where $\beta \in R^M$ and $M = \dim\{\mathrm{Null}[J_F(x_{fe})]\}$. Thus the goal for this step becomes: Find β which minimizes φ. Since the objective function, φ, involves the parameter alone only the last N equations of 4.8 require consideration. Thus

$$r_t = r_{fe} + V_r(x_{fe})\beta \tag{4.9}$$

characterizes the parameter part of x_t used in the objective function. Let γ be fixed. Note that:

$$\begin{aligned} \min_{x_t \in T_F(x_{fe})} \varphi(r_t, \gamma) &= \min_{x_t \in T_F(x_{fe})} \|r_t - r_\gamma\|_2 \qquad (4.10) \\ &= \min_{\beta \in R^M} \|D_\gamma(r_{fe} + V_r(x_{fe})\beta - r_\gamma)\|_2 \end{aligned}$$

But since $D_\gamma r_\gamma = D_\gamma r_o$ this become:

$$\min_{x_t \in T_F(x_{fe})} \varphi(r_t, \gamma) = \min_{\beta \in R^M} \|A_\gamma \beta - B_\gamma\|_2 \tag{4.11}$$

where

$$A_\gamma = D_\gamma V_r(x_{fe}) \quad \text{and} \quad B_\gamma = D_\gamma(r_o - r_{fe}) \tag{4.12}$$

For a given x_{fe} and γ the matrix A_γ and the vector B_γ are known. Thus equation 4.11 is a linear least squares problem readily solved via techniques discussed previously[7,8].

Let β_γ denote the vector which satisfies 4.11. To find the minimum of the objective function for all r_t and γ, compute β_γ for each γ and select the one which satisfies:

$$\min_{\gamma \in \Omega(N,n_f)} \|A_\gamma \beta_\gamma - B_\gamma\|_2 \tag{4.13}$$

As discussed earlier performing this computation for every possible γ is unnecessary. To eliminate some fault indices from consideration construct a set $\Gamma \subset \Omega(N,n_f)$ of candidate faults based on the following criterion:

(i) If the i'th entry of r_{fe} is sufficiently close to the nominal value for the i'th component (e.g. within 10%) then eliminate from consideration all faults containing component i.

(ii) If the j'th entry of r_{fe} has the greatest deviation from nominal then eliminate from consideration all faults which do not contain component j.

Instead of the minimization in 4.13 perform:

$$\min_{\gamma \in \Gamma} \|A_\gamma \beta_\gamma - B_\gamma\|_2 \tag{4.14}$$

The complete algorithm has the structure shown in Figure 4.

5. Limited Fault Algorithm Examples

The purpose of this section is to present two example problems which will illustrate the theory and limited fault algorithm developed in this chapter. The first example is based on the 12 parameter circuit of Figure 5. The solution algorithm is employed to analyze 16 randomly selected faults within this circuit. The second example is the 26 parameter circuit of Figure 1. Practical implementation of the fault diagnosis procedure for this circuit requires the use of sparse matrix techniques. Although the development of sparse matrix algorithms is beyond the scope of this report the example is included to demonstrate the feasibility of the algorithm for larger systems.

Example 5.1: Consider the circuit of Figure 5. The circuit input and outputs are:

$$u_1 = V_1 \qquad y_1 = I_{C_1} \qquad y_2 = V_o \tag{5.2}$$

The parameter definitions, nominal values and component transfer functions appear in Table 13.

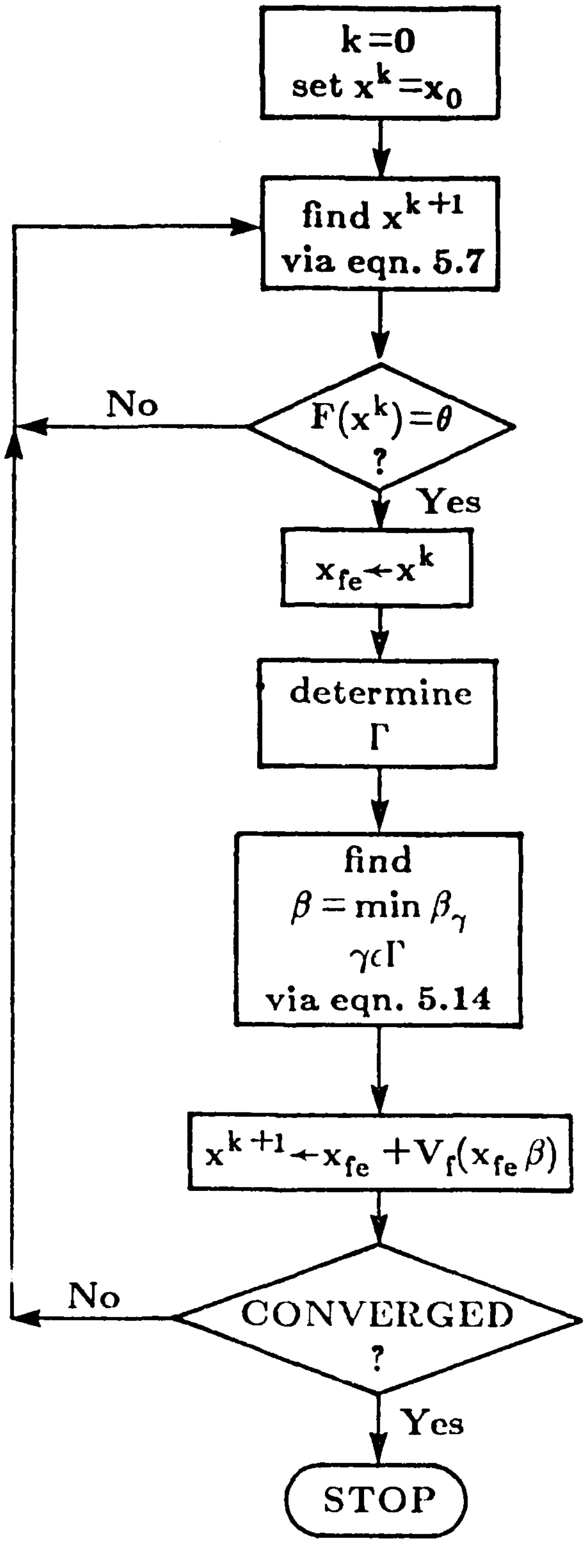

$k=0$
set $x^k = x_0$
find x^{k+1} via eqn. 5.7
$F(x^k)=\theta$?
No
Yes
$x_{fe} \leftarrow x^k$
determine Γ
find $\beta = \min_{\gamma \epsilon \Gamma} \beta_\gamma$ via eqn. 5.14
$x^{k+1} \leftarrow x_{fe} + V_f(x_{fe}\,\beta)$
CONVERGED ?
No
Yes
STOP

Figure 4. Detailed Algorithm Structure

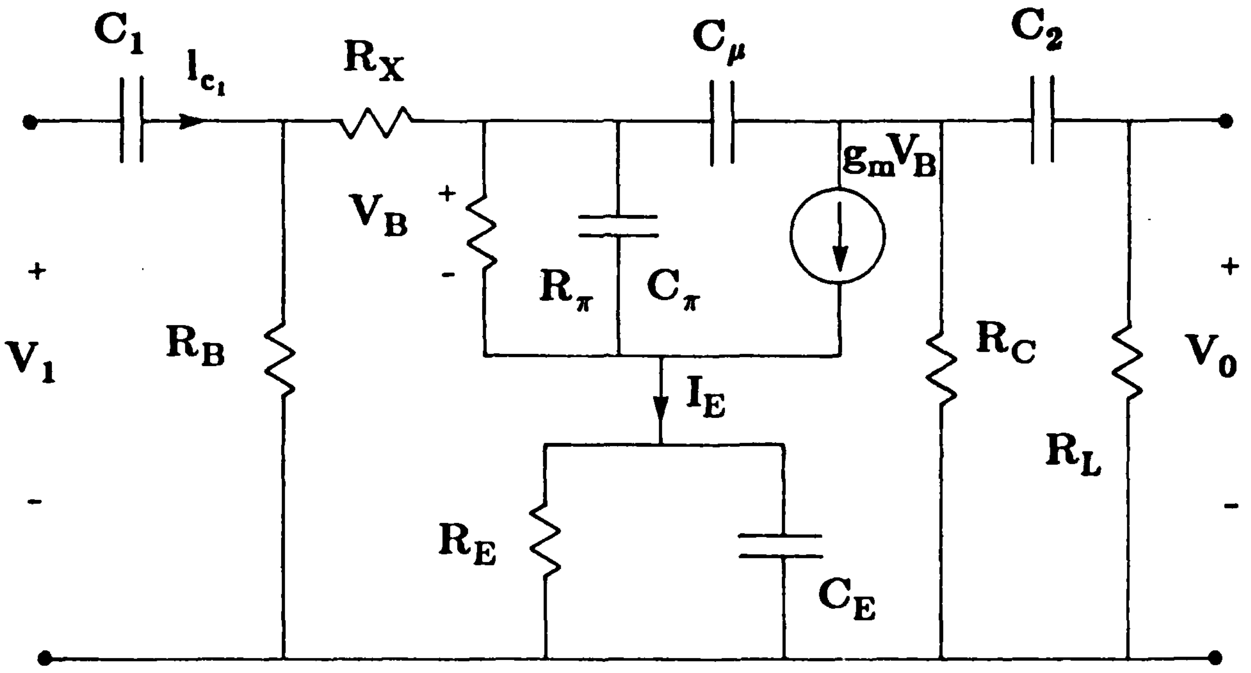


Figure 5. Fault Diagnosis Example (12 parameters).

Table 13.
Component information for Example 5.1

Parameter	Nominal Value	Definition	$Z_i(s,r_i)$
r_1	.1	$1/C_1$	r_1/s
r_2	.5	R_x	r_2
r_3	1.	R_π	r_3
r_4	1.	$1/C_\mu$	r_4/s
r_5	.1	$1/C_2$	r_5/s
r_6	1.	$1/R_B$	r_6
r_7	1.	$1/R_E$	r_7
r_8	1.	C_π	r_8s
r_9	.1	C_E	r_9s
r_{10}	1.	g_m	r_{10}
r_{11}	1.	$1/R_C$	r_{11}
r_{12}	.5	$1/R_L$	r_{12}

The nonzero entries of the sparse set of connection matrices, L_{11}, L_{12}, L_{21} and L_{22} appear in Table 14.

Table 14.
Nonzero entries of the connection matrices for Example 5.1.

L_{ij}	row,column	value	row,column	value	row,column	value
L_{11}	1,6	1	1,7	1	1,9	1
	1,11	1	1,12	1	2,7	1
	2,9	1	2,11	1	2,12	1
	3,7	1	3,8	-1	3,9	1
	3,10	-1	4,10	1	4,11	1
	4,12	1	5,12	1	6,1	-1
	7,1	-1	7,2	-1	7,3	-1
	8,3	1	9,1	-1	9,2	-1
	9,3	-1	10,3	1	11,1	-1
	11,2	-1	11,4	-1	12,1	-1
	12,2	-1	12,4	-1	12,5	-1
L_{12}	6,1	1	7,1	1	9,1	1
	11,1	1	12,1	1		
L_{21}	1,1	-1	1,2	-1	1,4	-1
	1,5	-1	2,6	1	2,7	1
	2,9	1	2,11	1	2,12	1
L_{22}	1,1	1				

The nonzero entries of L_{21}^R and V, computed via IMSL[8] routines LGINF and LLSQF respectively, appear in Table 15.

Table 15.
Nonzero entries of the L_{21}^{-R} and V matrices for Example 5.1.

matrix	row,column	value	row,column	value
L_{21}^{-R}	1,1	-0.25	2,1	-0.25
	4,1	-0.25	5,1	-0.25
	6,2	0.2	7,2	0.2
	9,2	0.2	11,2	0.2
	12,2	0.2		
V	1,1	0.866025	2,1	-0.288675
	2,2	-0.57735	2,3	-0.57735
	3,8	1	4,1	-0.288675
	4,2	0.788675	4,3	-0.211325
	5,1	-0.288675	5,2	-0.211325
	5,3	0.788675	6,4	0.861803
	6,5	-0.138197	6,6	-0.138197
	6,7	-0.138197	7,4	-0.138197
	7,5	0.861803	7,6	-0.138197
	7,7	-0.138197	8,9	1
	9,4	-0.138197	9,5	-0.138197
	9,6	-0.138197	9,7	0.861803
	10,10	1	11,4	-0.138197
	11,5	-0.138197	11,6	0.861803
	11,7	-0.138197	12,4	-0.447214
	12,5	-0.447214	12,6	-0.447214
	12,7	-0.447214		

For q=2, $s_1 = j10$ and $s_2 = j.6$ and $u(s_1) = u(s_2) = 1$ we compute the following nominal $\underline{\alpha}_i$:

$$\underline{\alpha}_1 = \begin{bmatrix} -.157584e+00 \\ -.333415e+00 \\ -.323898e+00 \\ .634977e+00 \\ .308373e-01 \\ .803794e-01 \\ -.290531e+00 \\ .398368c-01 \\ .384996e+00 \\ .398368e-01 \end{bmatrix} + j \begin{bmatrix} -.353887e-01 \\ -.130905e+00 \\ -.627419e-01 \\ -.121971e-01 \\ -.105105e+00 \\ -.732278e-01 \\ .359565e+00 \\ -.384996e-01 \\ .398368e+00 \\ -.384996e-01 \end{bmatrix} \qquad (5.3)$$

$$\underline{\alpha_2} = \begin{bmatrix} -.161031e+00 \\ .399337e+00 \\ -.247146e+00 \\ .705033e+00 \\ .275746e+00 \\ -.182022e+00 \\ -.178041e+00 \\ .213281e+00 \\ .252149e-01 \\ .213281e+00 \end{bmatrix} + j \begin{bmatrix} -.167796e+00 \\ -.395931e+00 \\ -.709381e-01 \\ .756794e-01 \\ -.126582e+00 \\ .153923e+00 \\ -.116167e+00 \\ -.420248e-01 \\ .127968e+00 \\ -.420248e-01 \end{bmatrix} \qquad (5.4)$$

This nominal data is used as a starting point for the solution algorithm.

The next step in the example is to simulate several 3-faults (3 parameters differ from nominal) and see if the solution algorithm works. Table 16 displays a set of 16 randomly selected fault indices and corresponding faulty parameter values. Each of these faults was simulated and the resulting measurement data applied to the solution algorithm. The final column in Table 16 indicates the performance of the algorithm for each fault. In 11 of the 16 cases the algorithm correctly identified all faulty parameters and determined their actual values. In four cases two of the three faults were correctly identified and in one case the algorithm selected as faulty a component which was actually good.

Table 16.
Fault list and algorithm performance summary for s_1=j10. and s_2=j.6.

Actual Fault Index	Faulty Parameter Values			Algorithm Fault Index
<6,7,10>	r_6=.5	r_7=2.	r_{10}=2.	<6,7,10>
<2,5,9>	r_2=1.	r_5=.05	r_9=.2	<2,9>
<1,4,9>	r_1=.2	r_4=.5	r_9=.2	<1,4,9>
<2,5,6>	r_2=.3	r_5=.07	r_6=1.4	<2,6>
<4,5,9>	r_4=2.	r_5=.05	r_9=.2	<4,5,9>
<2,6,11>	r_2=1.	r_6=.5	r_{11}=.5	<2,6,11>
<2,6,12>	r_2=1.	r_6=.5	r_{12}=.25	<2,6,12>
<7,8,9>	r_7=.5	r_8=2.	r_9=.2	<7,8,9>
<2,4,12>	r_2=.25	r_4=.5	r_{12}=.25	<2,4,12>
<5,6,8>	r_5=.05	r_6=.7	r_8=1.6	<6,8>
<2,8,10>	r_2=.25	r_8=2.	r_{10}=2.	<2,8,10>
<3,4,7>	r_3=.5	r_4=2.	r_7=2.	<3,4,7>
<3,6,9>	r_3=2.	r_6=.5	r_9=.2	<3,6,9>
<3,7,12>	r_3=2.	r_7=2.	r_{12}=.25	<3,7,12>
<2,4,5>	r_2=.3	r_4=1.5	r_5=.07	<2,4>
<5,7,12>	r_5=.05	r_7=2.	r_{12}=1.	<7,11,12>

Although the circuit is theoretically 3-fault diagnosable the results in Table 16 indicate that there are some practical problems associated with the determination of certain faults. For any given fault the most reliable indication of the theoretical capability to determine the fault is a test of the Jacobian evaluated at the fault but this is impractical since it is impossible to anticipate all possible faults. Instead the diagnosability test is based on of the rank of selected columns of the Jacobian, J_P, evaluated at the nominal point. Theoretically the results at the nominal point hold for almost all faults. Unfortunately it is quite possible for a matrix which is theoretically full rank to be less than full rank for a solution algorithm due to the finite word length of the computer.

A technique for circumventing this problem is to perform the diagnosis at several sets of test frequencies. Only those faults which are poorly conditioned at all test frequencies used would not be detectable. Suppose that the present example is repeated with the following test frequencies: $s_1 = j4.$ and $s_2 = j.2$. The resulting nominal $\underline{\alpha}_i$ are:

$$\underline{\alpha}_1 = \begin{bmatrix} -.152138e+00 \\ -.291921e+00 \\ -.309870e+00 \\ .639839e+00 \\ .411483e-01 \\ .892647e-01 \\ -.343296e+00 \\ .673079e-01 \\ .267029e+00 \\ .673079e-01 \end{bmatrix} + j \begin{bmatrix} -.306290e-01 \\ -.223071e+00 \\ -.464630e-01 \\ -.967588e-02 \\ -.825390e-01 \\ .328109e-01 \\ .961794e-01 \\ -.667572e-01 \\ .269232e+00 \\ -.667572e-01 \end{bmatrix} \tag{5.5}$$

$$\underline{\alpha}_2 = \begin{bmatrix} .522551e-01 \\ .550190e+00 \\ -.196403e+00 \\ .577404e+00 \\ .304301e+00 \\ -.246162e+00 \\ -.898594e-01 \\ .201163e+00 \\ -.157696e-01 \\ .201163e+00 \end{bmatrix} + j \begin{bmatrix} -.460309e+00 \\ .771991e-01 \\ -.890725e-01 \\ .307545e+00 \\ .939940e-01 \\ -.347141e-01 \\ -.883199e-01 \\ .788478e-01 \\ .402325e-01 \\ .788478e-01 \end{bmatrix} \tag{5.6}$$

The results of the diagnosis of the same 16 faults appears in Table 17.

Table 17.
Fault list and algorithm performance summary for $s_1=j4.$ and $s_2=j.2$.

Actual Fault Index	Faulty Parameter Values			Algorithm Fault Index
<6,7,10>	$r_6=.5$	$r_7=2.$	$r_{10}=2.$	<6,7,10>
<2,5,9>	$r_2=1.$	$r_5=.05$	$r_9=.2$	<2,5,9>
:1,4,9>	$r_1=.2$	$r_4=.5$	$r_9=.2$	<1,4,9>
<2,5,6>	$r_2=.3$	$r_5=.07$	$r_6=1.4$	<2,5,6>
<4,5,9>	$r_4=2.$	$r_5=.05$	$r_9=.2$	<4,5,9>
<2,6,11>	$r_2=1.$	$r_6=.5$	$r_{11}=.5$	<2,6,11>
<2,6,12>	$r_2=1.$	$r_6=.5$	$r_{12}=.25$	<2,6,12>
<7,8,9>	$r_7=.5$	$r_8=2.$	$r_9=.2$	<5,7,9>
<2,4,12>	$r_2=.25$	$r_4=.5$	$r_{12}=.25$	<2,4,12>
<5,6,8>	$r_5=.05$	$r_6=.7$	$r_8=1.6$	<5,6,8>
<2,8,10>	$r_2=.25$	$r_8=2.$	$r_{10}=2.$	<2,3,10>
<3,4,7>	$r_3=.5$	$r_4=2.$	$r_7=2.$	<3,4,7>
<3,6,9>	$r_3=2.$	$r_6=.5$	$r_9=.2$	<3,6,9>
<3,7,12>	$r_3=2.$	$r_7=2.$	$r_{12}=.25$	<3,7,12>
<2,4,5>	$r_2=.3$	$r_4=1.5$	$r_5=.07$	<2,4,5>
<5,7,12>	$r_5=.05$	$r_7=2.$	$r_{12}=1.$	<5,7,11>

A diagnosis based on the combination of the results of Tables 15 and 16 identifies ALL faulty components in the sixteen randomly selected fault combinations. The only diagnosis errors are three of the sixteen cases in which a good component was *identified as faulty*.

Example 5.7: Next reconsider the circuit of Figure 1. Due to the large size of this example a detailed analysis is not feasible without the utilization of sparse matrix techniques. Although the adaptation of the solution program to sparse matrices is beyond the scope of the current research a limited analysis of the example is included to demonstrate the feasibility of this approach for large circuits and to illustrate some of the properties of the algorithm.

The choice of inputs and outputs are denoted in the figure as u_i and y_i respectively. For this example N=26, $n_o=4$ and p=22. Notice that five of the seven accessible nodes are utilized to provide the single input and four outputs shown in Figure 1. Normally it would be desirable to utilize all accessible points to acquire the maximum of available information for the diagnosis procedure however we have elected to let $n_f=5$ and wish to illustrate the case where $n_o<n_f$.

The nonzero entries of the sparse set of connection matrices, L_{12} and L_{21}, appear in Table 18 and those of L_{11} appear in Table 1 (Chapter 1). All entries of L_{22} are zero. The computation of L_{21}^{-R} and V was performed via the IMSL[8] routines LGINF and LSVDF respectively. These too are spare matrices with their nonzero entries given in Table 18.

Table 18.
Nonzero entries of matrices for Example 5.7.

L_{ij}	row,column	value	row,column	value	row,column	value
L_{12}	1,1	1				
L_{21}	1,1	1	2,9	1	3,19	1
	3,26	1	4,22	1		
L_{21}^{-R}	1,1	1	9,2	1	19,3	0.5
	22,4	1	26,3	0.5		
V	2,2	1	3,3	1	4,4	1
	5,5	1	6,6	1	7,7	1
	8,8	1	10,9	1	11,10	1
	12,11	1	13,12	1	14,13	1
	15,14	1	16,15	1	17,16	1
	18,17	1	19,1	1	20,18	1
	21,19	1	23,20	1	24,21	1
	26,1	-1				

The component transfer functions are: $Z_i(s,r_i) = r_i s$ for i=4,5,10,11,16,17,23,24 and $Z_i(s,r_i) = r_i$ for the remaining i. The nominal component values as well as parameter values for three 5-faults appear in Table 19. Note that the component values are scaled to improve the numerical condition of the problem. The impedance scale factor is 10^2 and the frequency scale factor is 10^7.

Table 19.
Nominal parameter values and selected 5-faults.

component	units	nominal	fault 1 <4,13,18,20,22>	fault 2 <1,9,13,20,24>	fault 3 <3,6,14,17,25>
1	ohm	12	12	18	12
2	mho	0.1	0.1	0.1	0.1
3	ohm	56.7	56.7	56.7	35
4	farad	50	80	50	50
5	farad	5	5	5	5
6	mho	10	10	10	5
7	ohm	30	30	30	30
8	mho	0.1	0.1	0.1	0.1
9	ohm	10	10	3	10
10	farad	50	50	50	50
11	farad	5	5	5	5
12	mho	10	10	10	10
13	mho	0.3	0.2	0.5	0.3
14	ohm	10	10	10	15
15	ohm	2	2	2	2
16	farad	50	50	50	50
17	farad	5	5	5	8
18	mho	10	18	10	10
19	ohm	10	10	10	10
20	mho	0.3	0.5	0.5	0.3
21	ohm	10	10	10	10
22	ohm	10	6	10	10
23	farad	50	50	50	50
24	farad	5	5	3	5
25	mho	10	10	10	15
26	ohm	0.78	0.78	0.78	0.78

The number of test frequencies is q=2 and the inputs and test frequencies are:

$$u(s_1)=u(j.2)=2. \qquad u(s_1)=u(j.01)=.1 \qquad (5.8)$$

The nominal values for the $\underline{\alpha}_i$, i=1,2 appear in Table 20.

Table 20.
Nominal values for $\underline{\alpha}_1$ and $\underline{\alpha}_2$.

$\underline{\alpha}_1$ real part	$\underline{\alpha}_1$ imag. part	$\underline{\alpha}_2$ real part	$\underline{\alpha}_2$ imag. part
−.658870e+00	−.460866e+00	−.236072e+00	.357115e+00
−.956600e−04	−.380546e−03	.155763e−03	−.658132e−04
.160988e+01	−.272166e+01	.910666e+00	−.332734e+00
.380546e−01	−.956600e−02	.329066e−03	.778815e−03
.182796e+01	.237069e+00	.234965e−01	.270696e−01
−.956600e−02	−.380546e−01	.155763e−01	−.658132e−02
.312299e+00	−.125561e+01	.501926e+00	.106119e+00
.103920e−03	−.571410e−02	.830079e−03	.866948e−03
.571410e+00	.103920e−01	−.433474e−02	.415040e−02
.125561e+01	.312299e+00	−.530596e−02	.250963e−01
.103920e−01	−.571410e+00	.830079e−01	.866948e−01
.245900e+00	−.175069e+00	.434513e−01	.524036e−01
.770900e−02	−.193858e−01	.101418e−01	−.268636e−02
.543437e+00	−.237413e+00	.207551e+00	−.441227e−01
.193858e+00	.770900e−01	.134318e−02	.507090e−02
.512714e−01	−.250221e+00	.410939e−01	.476014e−01
.770900e−01	−.193858e+00	.101418e+00	−.268636e−01
.198924e+00	−.307626e+00	−.201038e−03	.265215e−01
.444493e−02	−.254349e−02	.357117e−02	.211851e−03
.254349e−01	.444493e−01	−.105925e−03	.178559e−02
.173045e+00	−.351821e+00	−.452230e−03	.247148e−01
.444493e−01	−.254349e−01	.357117e−01	.211851e−02

The results of algorithm for this example comprise Table 21. The program correctly identified faults 1 and 2 and four of five faulty components in fault 3. The difficulty in identifying parameter 14 is another example of a poorly conditioned problem however in this case the difficulty is not frequency dependent. The sensitivity of the four outputs to changes in parameter 14 is extremely small at all test frequencies. Under such a circumstance it is reasonable to expect difficulty in determining the parameter since the outputs contain little information about it.

Table 21.
Results for example 5.7.

fault #	actual γ	solution γ
1	<4,13,18,20,22>	<4,13,18,20,22>
2	<1,9,13,20,24>	<1,9,13,20,24>
3	<3,6,14,17,25>	<3,6,17,25>

This solution is a good example of the reduction in the number of fault combinations which results from the determination of a feasible point. For this example n_f=5 and N=26. Consequently there are $C_{26:5}$ or 65780 possible fault combinations. Thus a "brute force" approach to this example requires the solution of 65780 different sets of fault diagnosis equations. One the other hand the algorithm developed here finds a finds a feasible point first and then searches for the fault among the parameters which deviate significantly from nominal at the feasible point.

For example in the solution for fault 1 there were 10 parameters which deviated greater than $\pm 10\%$ from nominal. If the remaining components are considered good and the parameter with the greatest deviation from nominal is considered definitely bad then four faulty components remain to be found from nine possibilities. This means that the algorithm must test $C_{9:4}$ or 126 different faults. This is significantly fewer that 65780.

These examples employed a FORTRAN program based on the solution algorithm described in the previous section and compiled and executed on a VAX 11/780 computer. It is also noteworthy that both examples identify a reasonable number of simultaneous faults while achieving the goal, $n_o \leq \sqrt{N}$; set in [10].

6. Conclusions

Clearly the Tableau Approach is readily adaptable to the assumption of a limited number of simultaneous faults. The major problem associated with this assumption is the need to avoid testing the enormous number of fault combinations which occurs for large systems with more than a couple of faults possible. The algorithm developed here avoids this problem by utilizing the information available at the surface described by the equality constraints. The most significant aspect of this approach is that it does not require the number of test points to be greater than the number of assumed faults. This allows a reduction in test point requirements over other methods.

REFERENCES

[1] L. Rapisarda and R. DeCarlo, "Fault Diagnosis of Nonlinear Analog Circuits: Vol. II A Multifrequency Method For Soft Failure Analysis", Tech. Report TR-EE 82-22, Purdue University, W. Lafayette IN. ,March 1982.

[2] Z. F. Huang, C. Lin, and R. Liu, "Node-Fault Diagnosis and Design of Testability," *Proc. 20th IEEE Conf. on Decision and Control*, Vol. 3, pp. 1037-1042, 1981.

[3] C. -c. Wu, K. Nakajima, C. -L. Wey, and R. Saeks, "Analog Fault Diagnosis with Failure Bounds," *IEEE Trans. on Circuits and Systems*, Vol. CAS-29, pp. 277-284, May 1982.

[4] R. M. Biernacki and J. W. Bandler, "Multiple-Fault Location of Analog Circuits," *IEEE Trans. on Circuits and Systems*, Vol. CAS-28, 361-367, May 1981.

[5] R. A. DeCarlo and R. Saeks, *Interconnected Dynamical Systems*, New York: Marcel Dekker, 1981.

[6] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Reading: Addison-Wesley, 1973. Wesley,

[7] Gilbert Strang, *Linear Algebra and Its Application*, New York: Academic Press, 1976.

[8] IMSL - International Mathematical and Statistical Libraries, IMSL Inc., Houston.

[9] N. Navid and A. N. Willson, "A Theory and an Algorithm for Analog Circuit Fault Diagnosis," *IEEE Trans. on circuits and System*, Vol. CAS-26, pp. 440-457, July 1979.

[10] R. Saeks, "Criteria for Analog Fault Diagnosis," in *Proc. European Conf. Circuit Theory and Design*,(The Hague), Aug. 1981, 75-78.

APPENDIX A: Full Diagnosis Program

```
c       LATEST VERSION ------6 DEC 82----------------------
c       program driver(input,output,tape5=input,tape6=output)
c
c
c       This program solves the fault diagnosis equations under
c       the assumption that the equations are quadratic. Sparse
c       matrix techniques are not used. Since all matrices are
c       sparse, a sparse matrix implementation is advisable.
c
c       Development stage begun 21 April 1982.
c       Finished with completion of Chap5 on July 19, 1982.
c       Modified for CAS paper October 8,1982.
c       Modified to solve for normalized parameters December 6, 1982
c       Modified to use llbqf instead of llsqf December 6, 1982
c
c       This version uses the Newton-Raphson search direction but
c       uses the fact that the components of F(x) are quadratic to
c       interpolate ||F(x)||**2 along the search direction. Since
c       ||F(x)||**2 is fourth order a five point interpolation is
c       exact. The interpolant is used to find the point along the
c       search direction for which ||F(x)||**2 is minimized.
c
c
c
c       variable list:
c
c               integers
c
c       n       number of parameters, also row dimension of Z
c
c       q       number of test frequencies
c
c       p       dimension of the ambiguity
c
c       in      dimension of the input vector
c
c       ou      dimension of the output vector
c
c       zpow    power of s in Z(s), usually -1,0,+1
c
c       ip      integer work vector used by imsl routine llsqf
c
c       kb      used by imsl routine llsqf
c
c               real
c
```

```
c       l11     ccm connection matrix
c
c       l12     ccm connection matrix
c
c       l21     ccm connection matrix
c
c       l22     ccm connection matrix
c
c       l21r    right inverse of l21
c
c       v       matrix of basis vectors for the null space of l21
c
c       rnom    nominal parameter values
c
c       ract    actual parameter values
c
c       rkth    parameter values during the iteration process
c
c       rkthnm  rkth/rnom (normalized unknown)
c
c       dev     deviation from nominal for actual
c
c       a       real matrix equivalent to jf
c
c       ff      equivalent to f (see f)
c
c       dd      equivalent to d (see d)
c
c       xx      equivalent to x (see x)
c
c       tol     used by imsl routine llsqf
c
c       normsq  function name
c
c       lambda  function name
c
c       g       array of function values along the search direction
c
c       time1   beginning cpu time
c
c       time2   ending cpu time
c
c               complex
c
c       s       array of test frequencies (equiv. to ss)
c
c       u       array of input vectors (equiv. to uu)
c
c       f       right hand vector of algor. (equiv. to ff)
c
c       d       search direction (equiv. to dd)
c
c       x       point along search direction (equiv. to xx)
c
c       tmp     temporary storage
c
c       aa      work storage
```

```
c
c       ymnom   array of nominal test outputs (equiv. to yymnom)
c
c       alpha   array of nominal ambiguity vectors (equiv. to aalpha)
c
c       ymact   array of actual test outputs (equiv. to yymact)
c
c
c       Since the free-format read(5,*) won't read complex numbers
c       the variables s,u,ymnom,alpha,ymact are made equivalent to
c       real arrays of double size.  The reals are read but the
c       complex are used.
c
c
c
        logical convrg
        integer n,q,p,in,ou,zrow(30)
        integer lda,ldjf,ldl11,lda0,ldb0,ldal,ldv,ip(200),kb
        real l11(30,30),l12(30,5),l21(10,30),l22(10,5),l21r(30,10)
        real v(30,25),rnom(30),ract(30),a(250,250),ff(250),dd(250)
        real ss(10),uu(10,5),yymnom(20,5),aalpha(50,5),yymact(20,5)
        real normsq,lambda,tol,xx(250),g(5),dev(30)
        real time1,time2,rkth(30),rkthnm(30),c(4)
        complex s(5),u(5,5),ymnom(10,5),alpha(25,5),ymact(10,5)
        complex jf(150,250),d(125),f(125),b0(30,5),a0(30,5)
        complex tmp(200),aa(30,5),x(125)
        equivalence (ss,s),(uu,u),(aalpha,alpha)
        equivalence (yymact,ymact),(yymnom,ymnom)
        equivalence (ff,f),(dd,d),(xx,x)
c
c
        call second(time1)
c
c       set the row dimensions
c
        lda=250
        ldjf=150
        ldl11=30
        ldv=30
        ldal=25
        lda0=30
        ldb0=30
c
c       set precision of solution ie .01 = 1%
        prec = .001
c
c       Input secti r
c       read in from standard input
c
        read(5,*) n,q,p,in,ou
        read(5,*) (rnom(i),i=1,n)
        read(5,*) (ract(i),i=1,n)
        do 55 i=1,n
55      read(5,*)(l11(i,j),j=1,n)
        do 60 j=1,in
60      read(5,*)(l12(i,j),i=1,n)
        do 65 i=1,ou
```

```
65        read(5,*)(l21(i,j),j=1,n)
          do 70 j=1,in
70        read(5,*)(l22(i,j),i=1,ou)
          do 75 j=1,ou
75        read(5,*)(l21r(i,j),i=1,n)
          do 80 j=1,p
80        read(5,*)(v(i,j),i=1,n)
          read(5,*)(zpow(i),i=1,n)
          read(5,*)(ss(i),i=1,2*q)
          do 85 i=1,in
85        read(5,*)(uu(2*i-1,j),uu(2*i,j),j=1,q)
          do 90 i=1,ou
90        read(5,*)(yymnom(2*i-1,j),yymnom(2*i,j),j=1,q)
          do 95 i=1,p
95        read(5,*)(aalpha(2*i-1,j),aalpha(2*i,j),j=1,q)
          do 100 i=1,ou
100       read(5,*)(yymact(2*i-1,j),yymact(2*i,j),j=1,q)
c
          ngraph=0
          itmax=200
          do 98 i=1,p
98        read(5,*,end=101)
          read(5,*,end=101) ngraph, itmax
c
c
c         print a program heading
101       write(6,102)
102       format(//5x,38hFault Diagnosis Program-Tableau Method,/,
     &    10x,28h(Interpolate to find lambda))
          write(6,103)
103       format(/10x,18hNominal Parameters)
          write(6,107) ('r(',i,')=',rnom(i),i=1,n)
107       format(4(3x,a2,i2,a2,e10.4))
          write(6,110)
110       format(/18x,16hTest Frequencies)
          do 111 i=1,q
          write(6,112) i,s(i)
          write(6,113)(uu(j,i),j=1,2*in)
111       continue
112       format(12x,2hs(,i2,2h)=,e11.4,3h +j,e11.4)
113       format(12x,2hu=,10(f4.1,1x,f4.1,5x))
c
c
c         Compute the vector b0 = l21r(ymact-l22*u)
c
          do 130 i=1,q
          do 125 j=1,ou
          tmp(j)=cmplx(0.,0.)
          do 120 k=1,in
120       tmp(j)=tmp(j)+l22(j,k)*u(k,i)
125       tmp(j)=ymact(j,i)-tmp(j)
          do 130 j=1,n
          b0(j,i)=cmplx(0.,0.)
          do 130 k=1,ou
130       b0(j,i)=b0(j,i)+l21r(j,k)*tmp(k)
c
c
```

```
c        Compute the vector a0
c
         do 204 i=1,q
         do 204 j=1,n
         a0(j,i)=cmplx(0.,0.)
         do 201 k=1,n
201      a0(j,i)=a0(j,i)+l11(j,k)*b0(k,i)
         do 204 k=1,in
204      a0(j,i)=a0(j,i)+l12(j,k)*u(k,i)
c
c
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
         do 280 i=1,n
         rkthnm(i)=1.
280      dev(i)=(ract(i)-rnom(i))*100./rnom(i)
         iter=0
         xlam=-1.
290      iter=iter+1
         if(iter.gt.itmax) go to 900
c
         do 295 i=1,n
295      rkth(i)=rkthnm(i)*rnom(i)
c        evaluate f
         do 300 i=1,q
         call quadf(f(i*n-n+1),rkth,alpha(1,i),s(i),zpow,l11,ldl11,
     &            v,ldv,a0(1,i),b0(1,i),n,p,tmp)
300      continue
         if(normsq(ff,2*n*q).le.1.e-10) go to 900
c
c
c        stop because the algorithm appears stuck in a relative min.
         if(xlam.eq.0.) write(6,305)
305      format(10x,37hcaught in a point of relative minimum)
         if(xlam.eq.0.) stop
c
         call jacob(jf,ldjf,rkth,alpha,ldal,s,n,p,q,zpow,l11,ldl11,
     &            v,ldv,a0,lda0,b0,ldb0,tmp,aa)
c
         call cxtorl(a,lda,jf,ldjf,n,p,q)
c
cpar     since jacob computes [ partial f / partial rkth ]
c        the last n columns of the matrix a must be adjusted to get
c        [ partial f / partial rkthnm ]
         do 315 i=1,2*n*q
         do 315 j=1,n
315      a(i,2*p*q+j)=a(i,2*p*q+j)*rnom(j)
cpar
c        see imsldoc llsqf for meaning of tol
         tol=0
c        value for kb indicates a is assumed to have independent cols.
         kb=2*p*q+n
         kb=0
c        use llsqf to find the search direction
c        llsqf is from the imsl library
cxxx     call llsqf(a,lda,2*n*q,2*p*q+n,f,tol,kb,d,tmp,ip,ier)
cddd
         nf=1
```

```
        ind=1
        c(1)=0.
        c(2)=0.
        c(3)=1.
        call llbqf(a,lda,2*n*q,2*p*q+n,f,lda,nf,ind,c,d,lda,ip,jf,ier)
cddd
c       NOTE: the r part of d is normalized
c
c       option to skip the lambda determination and set it to -1
c
        if(iter.le.ngraph) go to 400
c
        do 319 j=1,q
        do 319 i=1,p
319     x((j-1)*p+i)=alpha(i,j)-0.0*d((j-1)*p+i)
        do 321 i=1,n
321     xx(2*p*q+i)=(rkthnm(i)-0.0*dd(2*p*q+i))*rnom(i)
c       the -0.0 is the lower end of the interpolation data
c       compute function |f|**2 at lambda=0. -.25 -.5 -.75 -1.
        do 339 ilam=1,5
        do 333 i=1,q
        call quadf(f(i*n-n+1),xx(2*p*q+1),x((i-1)*p+1),s(i),zrow,
     &          l11,ldl11,v,ldv,a0(1,i),b0(1,i),n,p,tmp)
333     continue
        g(ilam)=normsq(ff,2*n*q)
        do 329 j=1,q
        do 329 i=1,p
329     x((j-1)*p+i)=x((j-1)*p+i)-.25*d((j-1)*p+i)
        do 331 i=1,n
331     xx(2*p*q+i)=xx(2*p*q+i)-.25*dd(2*p*q+i)*rnom(i)
c       the -.25 is the increment between the interpolation data
339     continue
        xlam=lambda(g)
c
c       update the alpha's and r
c
400     continue
        do 410 j=1,q
        do 410 i=1,p
410     alpha(i,j)=alpha(i,j)+xlam*d((j-1)*p+i)
        convrg=.true.
        do 420 i=1,n
        if(abs(xlam*dd(2*p*q+i)).gt.prec) convrg=.false.
420     rkthnm(i)=rkthnm(i)+xlam*dd(2*p*q+i)
        if(convrg) go to 900
c       write the parameters as a progress report
        write(6,*) '              rank of a :',int(c(4))
        if(iter.gt.ngraph)then
                write(6,455) iter,xlam
        else
                write(6,456) iter,xlam
        endif
455     format(/5x,26hnormalized r at iteration ,i3,4x,
     &          8h(lambda=,e12.5,1h))
456     format(/5x,26hnormalized r at iteration ,i3,4x,
     &          8h(lambda=,e12.5,8h forced))
        write(6,457) (rkthnm(i),i=1,n)
```

```
457     format(5x,4e15.5)
        go to 290
c
c       write out final values
900     continue
        call second(time2)
        write(6,923) iter-1,time2-time1
923     format(/5x,i5,2x,16h iterations and ,e12.6,14h sec. required)
        write(6,925)
925     format(/5x,36hParameter Values at Solution Point  ,
     &          19h% dev. from nominal,17h  % error in sol.)
        do 950 i=1,n
927     format(10x,2hr(,i2,4h) = ,e11.4,18x,f10.2,5x,f10.2)
950     write(6,927) i,rkthnm(i)*rnom(i),dev(i),
     &           100.*(ract(i)-rkthnm(i)*rnom(i))/ract(i)
        stop
        end




c
c
c
        subroutine quadf(f,r,alph,s,zpow,l11,ldl11,v,ldv,a0,b0,n,p,tmp)
c
c
c       This routine computes the nonlinear function which corresponds
c       to the fault diagnosis equation at one test frequency.
c
c
c       argument list:
c
c       f         output: f = Z(s ,r)[L  Valph +a0(s )] - Valph -b0(s )
c                               i      11      i    i              i     i
c
c       r         input:  parameter vector
c
c       alph      input:  ambiguity vector
c
c       s         input:  test frequency
c
c       zpow      input:  exponent of s in Z
c
c       l11       input:  ccm connection matrix
c
c       ldl11     input:  row dimension of l11 in calling program
c
c       v         input:  null space basis of l21
c
c       ldv       input:  row dimension of v in calling program
c
c       a0        input:  particular solution for a
c
c       b0        input:  particular solution for b
```

```
c
c       n        input:  number of parameters & row dimension of Z
c
c       p        input:  number of columns in v
c
c       tmp      input:  temporary storage (at least 2*n)
c
c
        integer zrow(1),n,p,ldl11,ldv,i,j
        real r(1),l11(ldl11,1),v(ldv,1)
        complex alph(1),s,f(1),a0(1),b0(1),tmp(1)
c
c
        do 20 i=1,n
        tmp(i)=0.
        do 10 j=1,p
  10    tmp(i)=tmp(i)+v(i,j)*alph(j)
  20    f(i)=-tmp(i)-b0(i)
        do 40 i=1,n
        tmp(i+n)=0.
        do 30 j=1,n
  30    tmp(i+n)=tmp(i+n)+l11(i,j)*tmp(j)
        tmp(i+n)=tmp(i+n)+a0(i)
  40    f(i)=f(i)+r(i)*s**zrow(i)*tmp(i+n)
        return
        end




c
c
c
        subroutine jacob(jf,ldjf,r,alpha,ldal,s,n,p,q,zrow,l11,ldl11,
     &                        v,ldv,a0,lda0,b0,ldb0,tmp,aa)
c
c
c       This routine computes the fault diagnosis jacobian to be
c       used in the Newton-Raphson or other iteration schemes.
c
c
c       argument list
c
c       jf       output: the jacobian matrix
c
c       ldjf     input:  the row dimension of jf in the calling program
c
c       r        input:  parameter vector
c
c       alpha    input:  matrix of ambiguity vectors
c
c       ldal     input:  row dimension of alpha in the calling program
c
c       s        input:  array of test frequencies
c
```

```
c         n         input:   number of parameters & row dimension of Z
c
c         p         input:   number of columns in v
c
c         q         input:   number of test frequencies
c
c         zpow      input:   exponent of s in Z
c
c         l11       input:   ccm connection matrix
c
c         ldl11     input:   row dimension of l11 in calling program
c
c         v         input:   null space basis of l21
c
c         ldv       input:   row dimension of v in calling program
c
c         a0        input:   array of particular solutions for a
c
c         lda0      input:   row dimension of a0 in the calling program
c
c         b0        input:   array of particular solutions for b
c
c         ldb0      input:   row dimension of b0 in the calling program
c
c         tmp       input:   temporary storage (at least 2*n)
c
c         aa        input:   work array (at least n*q)
c
c
c         Note: For arrays alpha(i,j),a0(i,j),b0(i,j) & aa(i,j) the
c         subscript j means the data of this column corresponds to
c         test frequency s(j).
c
          integer zpow(lda0),n,p,q,ldjf,ldal,ldl11,ldv,lda0,ldb0
          integer i,j,k,jj
          real r(lda0),l11(ldl11,ldl11),v(ldv,ldal)
          complex jf(ldjf,ldjf),alpha(ldal,q),s(q),a0(lda0,q),b0(ldb0,q)
          complex tmp(ldjf),aa(n,q),sz
c
c
c
c         Compute the vector aa=L11*V*alpha+a0
c
          do 304 i=1,q
          do 301 j=1,n
          tmp(j)=cmplx(0.,0.)
          do 301 k=1,p
301       tmp(j)=tmp(j)+v(j,k)*alpha(k,i)
          do 304 j=1,n
          aa(j,i)=a0(j,i)
          do 304 k=1,n
304       aa(j,i)=aa(j,i)+l11(j,k)*tmp(k)
c
c
c         Compute the Jacobian JF(x)
c
          do 601 i=1,n*q
```

```
      do 601 j=1,q*p+n
601   jf(i,j)=cmplx(0.,0.)
      do 610 k=1,q
      do 605 i=1,n
c
c     Compute dfi/dr * si(alphai)
c
      sz=s(k)**zpow(i)
      jf(n*(k-1)+i,q*p+i)=aa(i,k)*sz
      do 605 j=1,p
      do 605 jj=1,n
c
c     Compute Z(si,r)*L11*V-V
c
605   jf(n*(k-1)+i,p*(k-1)+j)=
     & jf(n*(k-1)+i,p*(k-1)+j)+l11(i,jj)*v(jj,j)
      do 610 i=1,n
      do 610 j=1,p
      sz=s(k)**zpow(i)
610   jf(n*(k-1)+i,p*(k-1)+j)=jf(n*(k-1)+i,p*(k-1)+j)*r(i)*sz-v(i,j)
c
      return
      end




c
c
c
      subroutine cxtorl(a,lda,jf,ldjf,n,p,q)
c
c     This routine converts the complex jacobian into an
c     equivalent real matrix. Note: The conversion is somewhat
c     of a hybrid since the original unknown vector is part
c     complex (the alpha's) and part real (the r's).
c
c
c     argument list
c
c     a        output: the real equivalent of jf
c
c     lda      input:  the row dimension of a in the calling program
c
c     jf       input:  the complex matrix to be converted
c
c     ldjf     input:  the row dimension of jf in the calling program
c
c     n        input:  the dimension of r, the real sub-vector
c
c     p        input:  the dimension of each complex sub-vector
c
c     q        input:  the number of complex sub-vectors
c
c
```

```
      integer lda,ldjf,n,p,q,i,j
      real a(lda,1)
      complex jf(ldjf,1)
c
c
      do 820 i=1,n*q
      do 810 j=1,p*q
      a(2*i-1,2*j-1)=real(jf(i,j))
      a(2*i,2*j)=real(jf(i,j))
      a(2*i-1,2*j)=-aimag(jf(i,j))
810   a(2*i,2*j-1)=aimag(jf(i,j))
      do 820 j=1,n
      a(2*i-1,2*p*q+j)=real(jf(i,p*q+j))
820   a(2*i,2*p*q+j)=aimag(jf(i,p*q+j))
c
      return
      end




c
c
c
      real function normsq(f,n)
c
c
      integer n,i
      real f(n)
c
c
      normsq=0.
      do 10 i=1,n
 10   normsq=normsq+f(i)*f(i)
      return
      end
c
c




c
c
c
      real function lambda(g)
c
c     This program uses 5 points along a function g( ) to find the
c     point, lambda, at which the function is a minimum. The ordinate
c     points are assumed to be 0.0, -.25, -.5, -.75 and -1
c
c     If  g(x)=a1*x**4+a2*x**3+a3*x**2+a4*x+a5 then
c
```

```
c       | 0                0           0      0  1| |a1|    |g(-0.0)|
c       |3.90625e-3 -1.5625e-2  .0625 -.25  1| |a2|    |g(-.25)|
c       | .0625        -.125       .25    -.5   1|x|a3| = |g(-0.5)|
c       |.31640625  -.421875    .5625 -.75  1| |a4|    |g(-.75)|
c       |   1              -1          1     -1   1| |a5|    |g(-1.0)|
c
c                          vand * a = g
c
c       variable list
c
c                 real
c
c       g         input:  contains g(-0.0).....g(-1.0)
c
c       vand      data:   contains the vandermonde matrix
c
c       vand2     work:   scratch copy of vand to compute a
c                 note: It would be more efficient to use the inverse of
c                       vand here.
c
c       a         work:   coefficients of the interpolant
c                         becomes the coefficients of 1st derivative
c
c                 complex
c
c       z         work:   the zeroes of the 1st derivative
c
c                 subroutines
c
c       leqt1f  linear equation solver from imsl
c
c       zpolr   polynomial root finder
c
c
        integer i,j
        real vand(5,5),vand2(5,5),a(5),g(5),wk(10)
        complex z(3)
        data vand/0.,3.90625e-3,.0625,.31640625,1.,0.,-1.5625e-2,-.125,
     &  -.421875,-1.,0.,.0625,.25,.5625,1.,0.,-.25,-.5,-.75,-1.,
     &  1.,1.,1.,1.,1./
c       set fmin to the value of the function at the current point so that
c       only points representing a decrease from this value can be selected
        fmin=g(1)
        do 80 i=1,5
        do 80 j=1,5
 80     vand2(i,j)=vand(i,j)
        call leqt1f(vand2,1,5,5,g,0,wk,ier)
c
c       print the coefficients
        write(6,100)(5-i,g(i),i=1,5)
 100    format(//5(1ha,i1,1h=,e10.4,2x))
        iroot=0
c       find coef. of 1st derivative
        do 10 i=1,5
 10     a(i)=float(5-i)*g(i)
c       find zeros of 1st derivative
        call zpolr(a,3,z,ier)
```

```
c
c
          do 50 i=1,3
          if(cabs(z(i)).gt.1.e15) go to 30
          if(abs(aimag(z(i))).gt.1.e-5) go to 30
c-----
c         this prevents a search in the opposite direction
c         if(real(z(i)).gt.0.) go to 30
c-----
          flam=real(z(i))
          fmag=0.
          do 25 k=1,5
 25       fmag=fmag+g(k)*flam**(5-k)
          if(fmin.le.fmag) go to 30
          iroot=i
          fmin=fmag
 30       continue
 50       continue
c
          if(iroot.ne.0) go to 90
          if(g(1).lt.0.) then
c         if the interpolant is upside down try lambda=1.
               lambda=-1.
               return
          else
               lambda=0.
               write(6,99)
          endif
 99       format(5x,20hERROR: No Minimum !!)
 90       lambda=real(z(iroot))
          if(abs(lambda).gt.15.) then
                   lambda=-1.
                   write(6,*)'error: lambda too large'
          endif
          return
          end



          subroutine second(time)
          real time
          time=cputim()
          return
          end
```

APPENDIX B: Limited Fault Diagnosis Program

```
c       EXPERIMENTAL VERSION ------11 DEC 82----------------------
        program nfmain
c
c
c       This program solves the fault diagnosis equations under
c       the assumption that the equations are quadratic and that
c       the number of faults is limited.
c
c       Development stage begun December 6, 1982.
c       Based of the fault diagnosis program driver.f.
c
        logical convrg
        integer n, q, p, in, ou, zrow(30)
        integer lda, ldjf, ldl11, lda0, ldb0, ldal, ldv, ip(200)
        real l11(30,30), l12(30,5), l21(10,30), l22(10,5), l21r(30,10)
        real v(30,25), rnom(30), ract(30), a(250,250), ff(250), dd(250)
        real ss(10), uu(10,5), yymnom(20,5), aalpha(50,5), yymact(20,5)
        real normsq, lambda, xx(250), g(5), dev(30), beta(30)
        real time1, time2, rkth(30), rkthnm(30), c(4), anull(250,360)
        complex s(5), u(5,5), ymnom(10,5), alpha(25,5), ymact(10,5)
        complex jf(150,300), d(125), f(125), b0(30,5), a0(30,5)
        complex tmp(200), aa(30,5), x(125)
        equivalence (ss,s), (uu,u), (aalpha,alpha)
        equivalence (yymact,ymact), (yymnom,ymnom)
        equivalence (ff,f), (dd,d), (xx,x), (anull,jf)
c
        call second(time1)
c
c       set the row dimensions
c
        lda=250
        ldjf=150
        ldl11=30
        ldv=30
        ldal=25
        lda0=30
        ldb0=30
c
c       set precision of solution ie .01 = 1%
        prec = .0000001
c
c       Input section
c       read in from standard input
c
```

```
        read(5,*) n,q,p,in,ou
        read(5,*) (rnom(i),i=1,n)
        read(5,*) (ract(i),i=1,n)
        do 55 i=1,n
55      read(5,*)(l11(i,j),j=1,n)
        do 60 j=1,in
60      read(5,*)(l12(i,j),i=1,n)
        do 65 i=1,ou
65      read(5,*)(l21(i,j),j=1,n)
        do 70 j=1,in
70      read(5,*)(l22(i,j),i=1,ou)
        do 75 j=1,ou
75      read(5,*)(l21r(i,j),i=1,n)
        do 80 j=1,p
80      read(5,*)(v(i,j),i=1,n)
        read(5,*)(zpow(i),i=1,n)
        read(5,*)(ss(i),i=1,2*q)
        do 85 i=1,in
85      read(5,*)(uu(2*i-1,j),uu(2*i,j),j=1,q)
        do 90 i=1,ou
90      read(5,*)(yymnom(2*i-1,j),yymnom(2*i,j),j=1,q)
        do 95 i=1,p
95      read(5,*)(aalpha(2*i-1,j),aalpha(2*i,j),j=1,q)
        do 100 i=1,ou
100     read(5,*)(yymact(2*i-1,j),yymact(2*i,j),j=1,q)
c
        itmax=200
        do 98 i=1,p
98      read(5,*,end=101)
        naf=1
        error=.15
        read(5,*,end=101) naf,error
c
c
c       print a program heading
101     write(6,102)
102     format(//20x,'Fault Diagnosis Program-Tableau Method',/,
     &  25x,'(Interpolate to find lambda)')
        write(6,103)
103     format(/25x,'Nominal Parameters')
        write(6,107) ('r(',i,')=',rnom(i),i=1,n)
107     format(4(3x,a2,i2,a2,e10.4))
        write(6,110)
110     format(/25x,'Test Frequencies')
        do 111 i=1,q
111     write(6,112) i,s(i),(uu(j,i),j=1,2*in)
112     format(10x,'s(',i2,')=',e11.4,' +j',e11.4,
     &          2x,'u=',10(f4.1,' +j',f4.1,5x))
c
        write(6,*)'        # of allowable faults:',  naf
        write(6,*)'        allowable error:',  error
c
c       Compute the vector b0 = l21r(ymact-l22*u)
c
        do 130 i=1,q
        do 125 j=1,ou
        tmp(j)=cmplx(0.,0.)
```

```
      do 120 k=1,in
120   tmp(j)=tmp(j)+l22(j,k)*u(k,i)
125   tmp(j)=ymact(j,i)-tmp(j)
      do 130 j=1,n
      b0(j,i)=cmplx(0.,0.)
      do 130 k=1,ou
130   b0(j,i)=b0(j,i)+l21r(j,k)*tmp(k)
c
c
c     Compute the vector a0
c
      do 204 i=1,q
      do 204 j=1,n
      a0(j,i)=cmplx(0.,0.)
      do 201 k=1,n
201   a0(j,i)=a0(j,i)+l11(j,k)*b0(k,i)
      do 204 k=1,in
204   a0(j,i)=a0(j,i)+l12(j,k)*u(k,i)
c
c
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      do 280 i=1,n
      rkthnm(i)=1.
280   dev(i)=(ract(i)-rnom(i))*100./rnom(i)
      iter=0
      itnext=0
      xlam=-1.
      write(6,'(///)')
290   iter=iter+1
      if(iter.gt.itmax) then
             write(6,*)'      iteration limit exceeded'
             go to 900
      endif
c
      do 295 i=1,n
295   rkth(i)=rkthnm(i)*rnom(i)
c     evaluate f
      do 300 i=1,q
      call quadf(f(i*n-n+1),rkth,alpha(1,i),s(i),zpow,l11,ldl11,
     &       v,ldv,a0(1,i),b0(1,i),n,p,tmp)
300   continue
      fnorm=normsq(ff,2*n*q)
      write(6,*) '                        ITERATION ',iter,'   |F|',fnorm
      write(6,*)
      if(fnorm.le.1.e-10) go to 500
c
c
c     stop because the algorithm appears stuck in a relative min.
      if(xlam.eq.0.) write(6,305)
305   format(10x,37hcaught in a point of relative minimum)
      if(xlam.eq.0.) stop
c
      call jacob(jf,ldjf,rkth,alpha,ldal,s,n,p,q,zpow,l11,ldl11,
     &       v,ldv,a0,lda0,b0,ldb0,tmp,aa)
c
      call cxtorl(a,lda,jf,ldjf,n,p,q)
c
```

```
cpar    since jacob computes [ partial f / partial rkth ] the last n column
c       the matrix a must be adjusted to get [ partial f / partial rkthnm ]
        do 315 i=1,2*n*q
        do 315 j=1,n
315     a(i,2*p*q+j)=a(i,2*p*q+j)*rnom(j)
cpar
c       use llbqf to find the search direction
c       llbqf is from the imsl library
cddd
        nf=1
        ind=1
        c(1)=0.
        c(2)=1.e-5
        c(3)=1.
        call llbqf(a,lda,2*n*q,2*p*q+n,f,lda,nf,ind,c,d,lda,ip,jf,ier)
cddd
c       NOTE: the r part of d is normalized
c
        do 319 j=1,q
        do 319 i=1,p
319     x((j-1)*p+i)=alpha(i,j)-0.0*d((j-1)*p+i)
        do 321 i=1,n
321     xx(2*p*q+i)=(rkthnm(i)-0.0*dd(2*p*q+i))*rnom(i)
c       the -0.0 is the lower end of the interpolation data
c       compute function |f|**2 at lambda=0. -.25 -.5 -.75 -1.
        do 339 ilam=1,5
        do 333 i=1,q
        call quadf(f(i*n-n+1),xx(2*p*q+1),x((i-1)*p+1),s(i),zpow,
     &          lll,ldlll,v,ldv,a0(1,i),b0(1,i),n,p,tmp)
333     continue
        g(ilam)=normsq(ff,2*n*q)
        do 329 j=1,q
        do 329 i=1,p
329     x((j-1)*p+i)=x((j-1)*p+i)-.25*d((j-1)*p+i)
        do 331 i=1,n
331     xx(2*p*q+i)=xx(2*p*q+i)-.25*dd(2*p*q+i)*rnom(i)
c       the -.25 is the increment between the interpolation data
339     continue
        xlam=lambda(g)
c
c       update the alpha's and r
c
        do 410 j=1,q
        do 410 i=1,p
410     alpha(i,j)=alpha(i,j)+xlam*d((j-1)*p+i)
        convrg=.true.
        do 420 i=1,n
        if(abs(xlam*dd(2*p*q+i)).gt.prec) convrg=.false.
420     rkthnm(i)=rkthnm(i)+xlam*dd(2*p*q+i)
        if(convrg) go to 500
c       write the parameters as a progress report
        write(6,*) '              rank of a :',int(c(4))
        write(6,455) iter,xlam
455     format(10x,26hnormalized r at iteration ,i3,2x,
     &          8h(lambda=,e12.5,1h))
456     write(6,457) (rkthnm(i),i=1,n)
457     format(5x,4e15.5)
```

```
      write(6,'(///)')
      go to 290
c
c     at this point the iteration has found a "feasible solution"
500   if(itnext.eq.iter) go to 900
      do 510 i=1,n
510   rkth(i)=rkthnm(i)*rnom(i)
      call jacob(jf,ldjf,rkth,alpha,ldal,s,n,p,q,zpow,l11,ldl11,
     &         v,ldv,a0,lda0,b0,ldb0,tmp,aa)
      call cxtorl(a,lda,jf,ldjf,n,p,q)
      do 515 i=1,2*n*q
      do 515 j=1,n
515   a(i,2*p*q+j)=a(i,2*p*q+j)*rnom(j)
      call nulla(a,lda,2*n*q,2*p*q+n,anull,lda,nldim,dd,tmp)
ctesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttest
      call finder(anull(2*p*q+1,1),lda,n,nldim,rkthnm,naf,error,beta)
c     compute anull*beta and put it in d
      do 600 i=1,2*p*q+n
      dd(i)=0.
      do 600 j=1,nldim
600   dd(i)=dd(i)+anull(i,j)*beta(j)
c
      do 610 j=1,q
      do 610 i=1,p
610   alpha(i,j)=alpha(i,j)+d((j-1)*p+i)
      convrg=.true.
      do 620 i=1,n
      if(abs(dd(2*p*q+i)).gt.prec) convrg=.false.
620   rkthnm(i)=rkthnm(i)+dd(2*p*q+i)
      itnext=iter+1
      write(6,630) iter,beta(nldim+1)
c     beta(nldim+1) contains the best norm for r
630   format(10x,'normalized r at iteration ',i3,2x,
     &        '(null space step: rnorm=',e10.4,')')
      if(.not.convrg) go to 456
ctesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttest
c     write out final values
900   call second(time2)
      write(6,923) iter-1,time2-time1
923   format(/5x,i5,2x,16h iterations and ,e12.6,14h sec. required)
      write(6,925)
925   format(/5x,36hParameter Values at Solution Point ,
     &        19h% dev. from nominal,17h  % error in sol.)
      do 950 i=1,n
927   format(10x,2hr(,i2,4h) = ,e11.4,18x,f10.2,5x,f10.2)
950   write(6,927) i,rkthnm(i)*rnom(i),dev(i),
     &         100.*(ract(i)-rkthnm(i)*rnom(i))/ract(i)
      stop
      end
c




      subroutine finder(anull,lda,n,nldim,rkthnm,naf,err,beta)
```

```
c
        integer neql, nldim, n, ldr, ieql(30), ineql(30), iwk(30),
     &         isave(30), index(30)
        real rkthnm(1), diff(30), rdiff(30), rnull(30,30), r(30),
     &         bsave(30), anull(lda,1), beta(1), wk(2500), c(4)
c
        ldr=30
        error=err
        do 30 i=1,n
 30     diff(i)=1.-rkthnm(i)
c
c       determine which parameters are already at nominal
        ii=0
        do 100 i=1,n
        if(abs(diff(i)).le.error) then
                ii=ii+1
                ieql(ii)=i
        endif
100     continue
c
        neql=ii
        noneql=n-neql
c       set ineql
130     ii=0
        do 150 i=1,n
        do 140 j=1,neql
140     if(i.eq.ieql(j)) go to 150
        ii=ii+1
        ineql(ii)=i
150     continue
        write(6,*)'      possible faults',(ineql(i),i=1,ii)
c
        sumold=1.e+10
cnnn    icnold=n
c
        ijob=0
c       add a group of the possibly bad to the good (nadd=noneql-naf)
        nadd=noneql-naf
        if(nadd.lt.0) nadd=0
160     call incr(index,nadd,noneql,ijob)
czzz    write(6,*)'index',(index(i),i=1,nadd)
        if(ijob.eq.0) go to 240
c
        do 170 i=1,nadd
170     ieql(neql+i)=ineql(index(i))
        neqns=neql+nadd
        do 180 i=1,neqns
        rdiff(i)=diff(ieql(i))
        do 180 j=1,nldim
180     rnull(i,j)=anull(ieql(i),j)
c
        ind=1
        ncols=1
        c(1)=0.
        c(2)=1.e-5
        c(3)=1.
        call llbqf(rnull,ldr,neqns,nldim,rdiff,ldr,ncols,ind,c,
     &              beta, ldr, iwk, wk, k)
c
c        compute a trial value of r
         do 190 i=1,n
         r(i)=rkthnm(i)
         do 190 j=1,nldim
190      r(i)=r(i)+anull(i,j)*beta(j)
c
c        compute a performance index to evaluate the fault combination
         sum=0.
         do 210 i=1,neqns
         errxx=abs(r(ieql(i))-1.)
         sum=sum+errxx**2
210      continue
czzzz    write(6,*)'   sum:',sum
         sum=sum**.5
cxxxx    write(6,*)'              ieql',(ieql(i),i=1,neqns),'    sum:',sum
c
cvvvvvvvvvvvvvvvvv
c        reject a solution with negative parameters
         do 215 i=1,n
         if(r(i).lt.0.) go to 160
215      continue
cvvvvvvvvvvvvvvvvv
c        compare to all previous
         if(sum.lt.sumold) then
                 sumold=sum
                 do 220 i=1,neqns
220              isave(i)=ieql(i)
                 do 230 i=1,nldim
230              bsave(i)=beta(i)
         endif
         go to 160
240      continue
         do 250 i=1,nldim
250      beta(i)=bsave(i)
         beta(nldim+1)=sumold
         return
         end
c
c
         subroutine incr(index,ni,n,ijob)
c
c        This subroutine increments the index vector, index.
c        The vector index should list the indices of n items taken ni at
c        a time.
c
c        variable list
c
c        index   input/output:   The index vector
c
c        ni      input:          Order of index
c
c        n       input:          Order of full set
c
c        ijob    input/output:   Control on input    0=initialize
c                                                    1=increment
```

```
c                                     Flag on output      0=end
c
        integer index(ni),ni,n,ijob
c
        if((ni.eq.0).and.(ijob.eq.0)) then
                ijob=1
                return
        else if((ni.eq.0).and.(ijob.eq.1)) then
                ijob=0
                return
        else if(ijob.eq.0) then
                do 110 i=1,ni
 110            index(i)=i
                ijob=1
                return
        else if(ni.eq.n) then
                ijob=0
                return
        else
                index(ni)=index(ni)+1
                do 130 i=ni,2,-1
                if(index(i).gt.n-ni+i) then
                        index(i-1)=index(i-1)+1
                        index(i)=index(i-1)+1
                endif
 130            continue
                do 150 i=2,ni
 150            if(index(i).gt.n-ni+i) index(i)=index(i-1)+1
                if(index(1).gt.n-ni+1) ijob=0
                return
        endif
        end




        subroutine nulla(a,lda,nrows,ncols,anull,ldnull,nldim,s,wk)
c
c       This subroutine computes the nullspace of the matrix a.
c       Initially the nullspace dimension is set to ncols-nrows
c       if this is positive or zero otherwise. The routine then
c       checks the singular values. The nullspace dimension is
c       incremented for every singular value which is too small
c       relative to the largest(first).
c
c
c       variable list
c
c       a         input:  matrix for which nullspace is desired
c                         a is destroyed
c
c       lda       input:  row dimension of a
c
c       nrows     input:  number of rows in a
c
```

```
c       ncols    input:   number of columns in a
c
c       anull    output:  the columns of v span null[a]
c
c       ldnull   input:   row dimension of v
c
c       nldim    output:  dimension of null[a]
c
c       s        work:    work array of dimension min(nrows,ncols)
c                         containing the singular values
c
c       wk       work:    work array of dimension 2*max(nrows,ncols)
c
c
        integer lda,ldnull,nrows,ncols,nldim
        real a(lda,1),anull(ldnull,1),s(1),wk(1)
c
        call lsvdf(a,lda,nrows,ncols,b,0,0,s,wk,ier)
        nldim=ncols-nrows
        if(nldim.lt.0) nldim=0
        do 10 i=1,min0(nrows,ncols)
 10     if(s(i).le.1.e-5*s(1)) nldim=nldim+1
        do 20 i=1,ncols
        do 20 j=1,nldim
 20     anull(i,j)=a(i,ncols-nldim+j)
        return
        end
```

The following subroutine listed in Appendix A are also required

quadf
jacob
cxtorl
normsq
lambda
second

```
c
      do 601 i=1,n*q
```

END

FILMED

5-83

DTIC